AL/HR-TP-1994-0029

ONTOLOGY CAPTURE METHOD (IDEF 5)

Benjamin Peraketh, PhD
Christopher Menzel, PhD
Richard J. Mayer, PhD
Florence Fillion
Michael T. Futrell
Paula S. DeWitte, PhD
Madhavi Lingineni

KNOWLEDGE BASED SYSTEMS, INCORPORATED
2726 LONGMIRE
COLLEGE STATION, TEXAS 77845

HUMAN RESOURCES DIRECTORATE
LOGISTICS RESEARCH DIVISION
2698 G Street
Wright-Patterson Air Force Base, Ohio 45433-7604

October 1994

Interim Technical Paper for the Period March 1992 to September 1994

AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6573

NOTICES

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation, or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

The Public Affairs Office has reviewed this paper and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This paper has been reviewed and is approved for publication.

James C. McManus

JOANN M. SARTOR, Capt, USAF
Contract Monitor

BERTRAM W. CREAM, Chief
Logistics Research Division

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 1994	Interim March 1992 to September 1994

4. TITLE AND SUBTITLE
Ontology Capture Method (IDEF5)

5. FUNDING NUMBERS
C - F33615-90-C-0012
PE - 63106F
PR - 2940
TA - 01
WU - 08

6. AUTHOR(S)
Benjamin Peraketh, PhD
Christopher P. Menzel, PhD
Richard J. Mayer, PhD
Florence Fillion
Michael T. Futrell
Paula S. de Witte, PhD
Madhavi Lingineni

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Knowledge Based Systems Incorporated
1408 University Drive East
College Station, TX 77840

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Armstrong Laboratory
Human Resources Directorate
Logistics Research Division
2698 G Street
Wright-Patterson AFB, OH 45433-6573

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AL/HR-TP-1994-0029

11. SUPPLEMENTARY NOTES
Armstrong Laboratory Technical Monitor: Capt JoAnn M. Sartor, (513) 255-7775

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

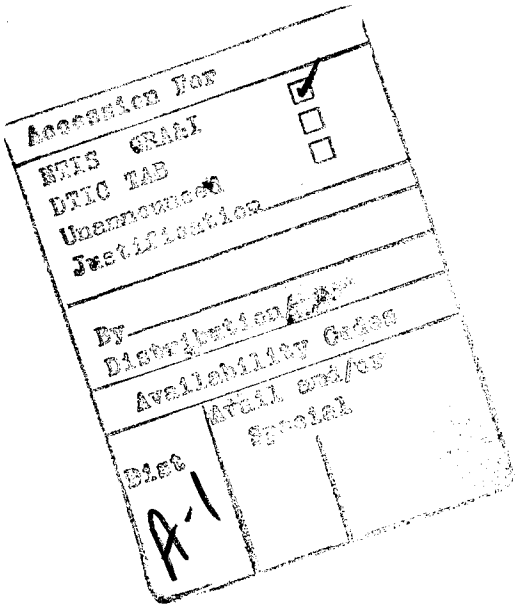
13. ABSTRACT *(Maximum 200 words)*

In order to exploit relevant information about a specific domain, the domain vocabulary must be captured. In addition, rigorous definitions of the basic terms in the vocabulary and the logical connections between those terms must be identified. Ontologies are used to capture the concepts and objects in a specific domain, along with associated relationships and meanings. In addition, ontology capture helps coordinate projects by standardizing terminology and creates opportunities for information reuse. The IDEF5 Ontology Capture Method has been developed to reliably construct ontologies in a way that closely reflects human understanding of the specific domain. IDEF5 relies on iterative knowledge extraction through various steps: organizing/scoping; data collection; data analysis; initial development; ontology refinement/validation. IDEF5 allows users to validate the vocabulary and axioms of a given domain and store that knowledge in a usable representational medium.

14. SUBJECT TERMS			15. NUMBER OF PAGES
ontology	Integration DEFinition	systems engineering	200
IDEF	knowledge acquisition	information systems	16. PRICE CODE
information engineering	method		

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

Table of Contents



List of Figures

Preface

This document provides a comprehensive description of the IDEF5 Ontology Description Capture Method. The IDEF5 method was developed under the Information Integration for Concurrent Engineering (IICE) project, F33615-90-C-0012, funded by Armstrong Laboratory, Logistics Research Division, Wright-Patterson Air Force Base, Ohio 45433, under the technical direction of Captain JoAnn Sartor and Mr. James McManus. The prime contractor for IICE is Knowledge Based Systems, Inc. (KBSI), College Station, Texas. The authors wish to acknowledge and extend special thanks to the following people who helped compose this document:

Julie Holden
James MacDougall
Richard McGuire

1 Executive Summary

This document provides a comprehensive description of the IDEF5 Ontology Capture Method. Its purpose is to guide a person in becoming proficient in applying IDEF5 to develop and manage domain ontologies effectively.

The IDEF5 Method Report is designed for the following audience:

- Knowledge engineers and application domain experts interested in developing, documenting, storing, and sharing domain knowledge;
- System analysts and designers interested in managing ontology knowledge effectively for the purposes of both analysis and design; and
- Researchers in the application of knowledge representation methods to problems in engineering and manufacturing.

The document is divided into the following four sections and two appendices:

1. An Executive Summary, which discusses the scope and content of the ontology capture method report and provides an initial overview of the method (Section 1);
2. A section on the conceptual foundations of ontology (Section 2);
3. A section on the ontology representation languages: a graphical language for expressing basic ontology information and a much richer, linear language (the "elaboration language") for expressing detailed ontology information (Section 4);
4. A section on the IDEF5 ontology development procedure, which discusses the application of the method for capturing and maintaining ontology information (Section 3);
5. An appendix containing the IDEF5 relation library, consisting of a collection of detailed characterizations of common domain relations (Appendix A); and
6. An appendix containing the formal Backus-Naur Form (BNF) for the elaboration language (Appendix B).

The authors anticipate the use of this document for a wide variety of purposes. Thus, the material is presented in a manner that allows readers to obtain the needed knowledge without having to read the entire document. The following guidelines are suggested for the use of this document.

- For an *executive overview*, read Sections 1 and 2.
- To become *proficient* in the development of accurate IDEF5 ontology descriptions, read the entire manual, with special emphasis on Sections 3 and 4.
- An IDEF5 *project leader* should study Section 3 in detail, but must also have an understanding of the method in its entirety. The introduction (Section 2) describes the motivations and potential uses for the IDEF5 method. Section 3 describes a procedure for developing IDEF5 ontology descriptions. Finally, Section 4 describes the IDEF5 ontology language.

1.1 Motivations

We characterize the meaning of the term "ontology" to include a catalog of terms used in a domain, the rules governing how those terms can be combined to make valid statements about situations in that domain, and the "sanctioned inferences" that can be made when such statements are used in that domain. In every domain, there are phenomena that the humans in that domain discriminate as (conceptual or physical) objects, associations, and situations. Through various language mechanisms, we associate definite descriptors (e.g., names, noun phrases, etc.) to that phenomena. In the context of "ontology," we use the term "relation" to refer to a definite descriptor that refers to an association in the real world. We use the term "term" to refer to a definite descriptor that refers to an object or situation-like thing in the real world. In an ontology, we try to catalog the descriptors (like a data dictionary) and create a model of the domain, if described with those descriptors. Thus, in building an ontology, you must produce three products. You have to catalog the terms, capture the constraints that govern how those terms can be used to make descriptive statements about the domain, and then build a model that when provided with a specific descriptive statement, can generate the "appropriate" additional descriptive statements. By appropriate descriptive statements we mean (i) because there are generally a large number of possible statements that could be generated, the model generates only that subset which is "useful" in the context, and (ii) the descriptive statements that are generated represent facts or beliefs that would be held by an intelligent agent in the domain who had received the same information. The model is then said to embody the "sanctioned inferences" in the domain. It is also said to "characterize" the behavior of objects and associations in the domain. Thus, an

ontology is similar to the now familiar data-dictionary, plus a grammar, plus a model of the behavior of the domain.

Another characterization of the meaning of "ontology" is given in the following excerpt from Tom Gruber (see [Gruber 93] also):

> The word "ontology" seems to generate a lot of controversy in discussions about AI. It has a long history in philosophy, in which it refers to the subject of existence. It is also often confused with epistemology, which is about knowledge and knowing.
>
> In the context of knowledge sharing, I use the term ontology to mean a specification of a conceptualization. That is, an ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents. This definition is consistent with the usage of ontology as set-of-concept-definitions, but more general. And it is certainly a different sense of the word than its use in philosophy.
>
> Ontologies are often equated with taxonomic hierarchies of classes, class definitions and the subsumption relation, but ontologies need not be limited to these forms. Ontologies are also not limited to conservative definitions, that is, definitions in the traditional logic sense that only introduce terminology and do not add any knowledge about the world (Enderton, 1972). To specify a conceptualization, one needs to state axioms that do constrain the possible interpretations for the defined terms.
>
> Pragmatically, a common ontology defines the vocabulary with which queries and assertions are exchanged among agents. Ontological commitments are agreements to use the shared vocabulary in a coherent and consistent manner. The agents sharing a vocabulary need not share a knowledge base; each knows things the other does not, and an agent that commits to an ontology is not required to answer all queries that can be formulated in the shared vocabulary.

Any domain with a determinate subject matter has its own terminology, a distinctive vocabulary that is used to talk about the characteristic objects and processes that comprise the domain. A library, for example, involves its own vocabulary having to do with books, reference items, bibliographies, journals, and so forth. Similarly, semiconductor manufacturing has its own language of chips, wafers, etchants, designs, and so on. The nature of a given domain is thus

revealed in the language used to talk about it. Clearly, however, the nature of a domain is not revealed in its corresponding vocabulary *alone*; in addition, one must (i) provide rigorous definitions of the grammar governing the way terms in the vocabulary can be combined to form statements and (ii) clarify the logical connections between such statements. Only when this additional information is available is it possible to understand both the natures of the individuals that exist in the domain and the critical relations they bear to one another. An *ontology* is a structured representation of this information. More exactly, an ontology is a domain vocabulary together with a set of precise definitions, or *axioms*, that constrain the meanings of the terms in that vocabulary sufficiently to enable consistent interpretation of statements that use that vocabulary.

Taken by itself, it may seem that there is not much difference between an ontology and a data dictionary. However, a data dictionary is typically just a compendium of terms together with definitions for the individual terms stated in natural language. By contrast, the grammar and axioms of an ontology are stated in a precise formal language with a very precise syntax and a clear formal semantics (see Section 4.2). Consequently, ontologies are, in general, far more rigorous and precise in their content than a typical data dictionary (and, hence, more so than a typical data "encyclopedia," because an encyclopedia is just a collection of related data dictionaries). Ontologies also tend to be more complete as well: *relations* between concepts and objects in a domain, and constraints on and between domain objects, are made explicit rather than left implicit, thus minimizing the chance of misunderstanding logical connections within the domain. A data dictionary, by contrast, generally relies upon an intuitive understanding of the terms in question and the logical connections between the concepts and objects they stand for. This works well enough in small restricted domains. But when information systems span organizational, geographic, and enterprise boundaries, problems arise. The traditional approach is problematic for several reasons, not the least of which is that different persons in different domains might understand the same term subtly different but important ways that are not uncovered in a natural language definition (which can lead to inconsistent interpretations of the same term across different contexts), and so forth. The regimentation of an ontology of the involved domains in a canonical language helps to avoid this problem. Furthermore, the discipline of expressing the ontology information in a formal language enhances the skills necessary for extracting the information, in particular, the ability to abstract from particular objects to the kinds of which they are instances, from particular connections to the relations such instances stand in generally, and from particular behaviors to the constraints that bind instances of various kinds together logically within the domain.

1.1.1 Motivations for Ontology

The ability to fix a domain vocabulary and its meaning in the context of use in this manner is critical for true concurrent engineering. A large engineering or manufacturing project involves the resources of many different clusters of cooperative agents (human or otherwise) in the given endeavor. Each cluster makes its own contributions, and the overall success of the project depends in large measure on the degree of integration between those different clusters throughout the development process. A key to effective integration is the accessibility of rich ontologies characterizing each of the domains addressed by each cluster. For instance, access to a manufacturing ontology that includes constraints on how a given part is manufactured can aid designers in their design of a complex product by giving them insight into the manufacturing implications of their design concepts. Similarly, access to an engineering ontology that includes constraints on how a given part is to function given a particular shape or fit can aid process planners in their development of the appropriate manufacturing processes. A commonly accessible collection of relevant ontologies thus permits more efficient sharing of information arising from various sources within the enterprise.

A related motivation for ontology capture is the *standardization* of terminology. An enormous problem in the coordination of large projects is the diversity of backgrounds the various kinds of engineers bring to their respective roles. As a consequence, many engineers use similar terminology in many different ways with many different connotations. Because of such differences, the information that one engineer *intends* to convey to another may in fact become garbled; in the best case, such miscommunications can be responsible for a great deal of lost time and resources; in the worst case, such miscommunications can result in the loss of life. Consequently, it is often necessary in the course of a large project to standardize the relevant vocabulary. The ontology capture method provides a principled method for carrying out this task efficiently and effectively, and maintaining the results of the task in a robust, accessible form.

This suggests another, strong motivation for ontology: reusability. Among the most significant problems in engineering and manufacturing in general is the redundant effort expended in capturing or recreating information that has already been recorded elsewhere. For example, in programming, the same kinds of routines (e.g., in the design of user interfaces) are often used again and again in different programs by (in general) different programmers. Consequently, enormous amounts of time and effort have gone into reinventing the wheel time and again. Recognition of this problem has led to the development of vast *libraries* that contain often used routines which programmers can simply call straight into their programs, rather than having to duplicate the function of existing code. Engineering and manufacturing face the same type of problem. Manufacturing domains, for

example, share many common features that are independent of the specific characteristics of a given domain; and the more similar the domains, the more such features they share. Rather than encoding this information all over again in every new setting, analogous of the concept of a programming library, one can imagine collecting this common information into *ontology libraries*, (i.e., large revisable ontology databases of structured, domain-specific information.) Information in these ontologies can then be reused and modified to suit the needs of the moment. Moreover, because ontologies provide a standardized terminology by their very nature, no special additional effort need be expended on fixing domain terminology. It must be emphasized, however, that, despite the potential size of a given ontology, the concept itself is highly *scaleable*; that is, ontologies are no less effective in smaller contexts than on very large ones.

1.1.2 Motivations for an Ontology Development Method

There is a global vision behind the idea of ontology development. Spearheaded by the Knowledge Sharing Effort sponsored by the Advanced Research Projects Agency (ARPA), ontologies are being constructed for a growing number of manufacturing, engineering, and scientific domains. With such ontologies in place, the advantages noted above could be realized on a global scale: standardized terminology with precise meanings that are fixed across industries and across international borders, and the ability to access and reuse a huge number of existing ontologies in the design and construction of new systems. Central products of this effort include the Knowledge Interchange Format (KIF), a text-based logical language for the interchange of knowledge, and Ontolingua, a mechanisms built on KIF for translating knowledge between different representation languages. The IDEF5 method described in this report has been designed with the Knowledge Sharing Effort and its vision closely in mind. Most notably, the IDEF5 elaboration language (see Section 4.2) — the central medium for storing ontology information collected via the IDEF5 method (see Section 3) — uses KIF as its foundation, and is thus wholly compatible with the central tools of the Knowledge Sharing Effort. This is particularly crucial as the concepts behind the effort become even more widely accepted and implemented.

Another key motivation for an ontology development method is a pragmatic one. Previous approaches to ontology have almost exclusively been academic in nature. Researchers from varied fields such as Artificial Intelligence, Philosophy, Database Management, Mathematics, and Cognitive Science have studied ontology from different perspectives. All previous approaches have failed to produce a practical method for ontology acquisition. The IDEF5 method therefore fulfills an important need by providing a cost-effective mechanism to acquire, store, and maintain scaleable and re-usable ontologies. The intended contribution of IDEF5 is a method to guide and assist domain experts and knowledge engineers in the construction of both small and large reusable

ontologies. We have designed the IDEF5 technique to be usable by a personnel at varying skill levels and from a variety of different kinds of organizations.

1.2 Benefits of Ontology Development

Ontology development provides several benefits to organized enterprises. The benefits of ontology development can be grouped under two headings:

1) Benefits of developing the ontology: The process of ontological analysis is a discovery process that leads to an enhanced understanding of a domain. The insights of ontological analysis are useful for (i) identification of problems (diagnosis), (ii) identification of the problem causes (causal analysis), (iii) identification of alternative solutions (discovery and design), (iv) consensus and team building, and (v) knowledge sharing and reuse.

2) Benefits derived from the products of ontology development: The ontologies that result at the end of an ontology development effort can be used beneficially for (i) information systems development: ontologies provide a blueprint for developing more intelligent and integrated information systems, (ii) systems development: ontologies can be used as reference models for planning, coordinating, and controlling complex product/process development activities, (iii) business process re-engineering: ontologies provide clues to identifying focus areas for organizational restructuring and suggest potential high-impact transition paths for restructuring.

Ontological analysis and development have been shown to be useful for: (i) Consensus building, (ii) Object-oriented design and programming, (iii) Component based programming, (iv) User interface design, (v) Enterprise information modeling, (vi) Business process reengineering, and (vii) Conceptual schema design.

1.3 Overview of the Report

The sections following this executive summary jointly constitute a comprehensive report on the IDEF5 ontology capture method. Section 2 of the report provides a detailed discussion of the conceptual foundations of IDEF5. It begins by tracing the roots of ontological inquiry, with respect to engineering and manufacturing, to the classical philosophical tradition — also known as "ontology" — of characterizing and classifying what ultimately exists. From this tradition springs the central concepts of ontology: kinds (roughly, classes or types), properties, attributes, relations, parts and wholes, and processes. Most of Section 2 is devoted to explicating and

relating these concepts. The section closes with a discussion of the need for a separate ontology method distinct from existing methods. (The relation of the ontology capture method with existing IDEF methods is discussed in Subsection 1.3.)

Section 3 provides a practical method for the construction of ontologies. Ontology development requires extensive iterations, discussions, reviews, and introspection. Knowledge extraction is usually a discovery process and requires considerable introspection. It requires a process that incorporates both significant expert involvement as well as the dynamics of a group effort. Given the open-ended nature of ontological analyses, it is not prudent to adopt a "cookbook" approach to ontology development. We recommend the use of a general procedure along with a set of useful guidelines. Section 3 describes the mechanics of such a process for potential IDEF5 ontology developers.

In brief, the IDEF5 ontology development process consists of the following five activities.

1. **Organizing and Scoping** This activity involves establishing the purpose, viewpoint, and context for the ontology development project and assigning roles to the team members.

2. **Data Collection** This activity involves acquiring the raw data needed for ontology development.

3. **Data Analysis** This activity involves analyzing the data to facilitate ontology extraction.

4. **Initial Ontology Development** This activity involves developing a preliminary ontology from the acquired data.

5. **Ontology Refinement and Validation** This activity involves refining and validating the ontology to complete the development process.

Although the above activities are listed sequentially, there is a significant amount of overlap and iteration between the activities. These activities, and their interconnections, are described in detail in Section 3.

Section 4 contains a description of the IDEF5 ontology languages. There are two such languages: the IDEF5 *schematic language* and the IDEF5 *elaboration language*. The schematic language is a graphical language that has been specially tailored to enable domain experts to express the most common forms of ontological information, especially with the aid of an automated ontology capture tool. This enables average users both to input the basic information needed for a first-cut ontology and to augment or revise existing ontologies with new information.

There is a price for the relative ease of use of the schematic language, viz., that it lacks the full expressive power needed to capture general ontology information. To capture such information is the purpose of the IDEF elaboration language. The elaboration language is a structured text language with the full expressive power of first-order logic and set theory. This enables a user to express virtually *any* condition, or relation, or fact that one might need to express to characterize a given kind of thing, or property, or relation, or process found in a domain. In addition to set theoretic constructs, the language also includes specialized constructs for expressing ontology information in the particular format of IDEF5. This makes for easy translation from the schematic language into the elaboration language, and vice versa, insofar as that is possible.

Finally, the report concludes with two appendices. The first is the (current) IDEF5 relation library of reusable ontology elements. This library is a rich repository of information consisting of a set of characterizations of (i.e., definitions and axioms for) commonly used relations. It provides a repository of formally defined and characterized relations that can be reused and customized in a particular project. The relation library itself is a specialized ontology: an ontology of commonly used relations. The motivation for this library grew out of the previously mentioned analogy with software engineering. Often in software development, the same kinds of routines are used again and again in different programs by (in general) different programmers. The development of ontologies will face the same sort of problem. It is likely that the same or similar relations will appear in a number of different ontologies. The role of a library of relations such as the one presented in Appendix A will be to enable modelers to reuse and customize relations that have been defined in previously captured ontologies. The library can also be used as a reference for the different ways to define and characterize relations and illustrative examples of the use of the IDEF5 elaboration language. All definitions and characterizing axioms in the library have been written using the IDEF5 elaboration language. Thus, the library can also serve as a useful learning tool for mastering the IDEF5 elaboration language. Finally, the library is extensible in that any relation that has been formally defined and characterized may be added to it.

The second appendix consists of the BNF specification of the IDEF5 elaboration language to ensure that the language is well-defined. A glossary for the report follows this appendix.

1.4 The Connection Between IDEF5 and Other Methods

As Mr. John Zachman in his seminal work on information systems architecture observed, "... there is not *an* architecture, but a *set* of architectural representations. One is not right and another wrong. The architectures are different. They are additive, complementary. There are reasons for electing to expend the resources for developing each architectural representation. And, there are

risks associated with *not* developing any one of the architectural representations." [Zachman 87] Consistent, reliable creation of correct architectural representations, whether artificial approximations of a system (models) or purely descriptive representations, requires the use of a guiding method. These observations underscore the need for *many* "architectural representations," and correspondingly, many methods.

Typically, methods, and their associated architectural representations, focus on a limited set of system characteristics and explicitly ignore those that are not directly pertinent to the task at hand. Thus, IDEFØ provides a compact, yet surprisingly powerful, conceptual universe for modeling business activities; for all its power, however, it would be highly inconvenient, if possible at all, to use it to design a relational database; IDEF1X is the method that is optimized for that task. Similarly, IDEFØ explicitly excludes temporal information, and limits what can be represented about temporal relations that hold between business activities, as well as the objects involved in the internal structure of those activities. These exclusions are what give IDEFØ its power in modeling business activities. For in a method design as in a programming language design, what distinguishes a well designed effective method is what is left out more so than what is left in. IDEF3, on the other hand, includes explicit representations of processes, time intervals, and temporal relations and, hence, is ideally suited for expressing information about timing and sequencing; it also includes the capacity to express arbitrary information about the individuals participating in those processes. It lacks, however, the specialized representations of IDEFØ and, therefore, information that IDEFØ expresses with great ease and simplicity is, by comparison, expressed only awkwardly in IDEF3.

The connection between these methods and IDEF5 is rather straightforward. Of the methods just mentioned, the IDEF5 schematic language is perhaps closest to IDEF1 and IDEF1X. However, the connection between IDEF1/1X and IDEF5 is analogous to that between IDEFØ and IDEF3. The information in an IDEF1 or IDEF1X model could in principle be expressed in the IDEF5 elaboration language. However, because it does not contain the well-designed, specialized representations of IDEF1/1X, it would be exceedingly cumbersome in IDEF5 to design a relational database, for example. But the expressive power of IDEF1/1X soon reaches its limits and, hence, could not possibly do all that is expected of a general ontology language. (For a more detailed comparison of IDEF1/1X and IDEF5, see Subsection 2.4.)

In a sense, the designs of both IDEF3 and IDEF5 break the traditional mold according to which methods are purposely designed with limited expressive power. The elaboration languages of both methods are full first-order languages (and more besides) and, hence, are capable of expressing most any information that might need to be recorded in a given domain. This break with tradition

not only reflects the need for greater expressive power, but also reflects the development and increased utilization of more intelligent tools and automated, model-driven systems in business and engineering. Intelligent tools and model-driven systems generally must manipulate much richer forms of information than can be expressed in a traditional method. This motivates the design of richer methods that have the capacity to represent and organize such information, methods that are not restricted to pencil and paper form and, hence, which truly augment the ability of human agents to create, manage, and reuse a richer store of knowledge. For the reasons above, these newer methods will *not* make the older, more restricted methods obsolete; the ability to filter and structure information relative to certain well-defined tasks will still be very useful. At the same time, the greater demands of intelligent tools and model-driven systems will require more.

The broader vision that guides these newer methods is one in which all system definition information is stored in a global (albeit perhaps *virtual*) repository of information, with modeling methodologies providing different views that filter the information in various useful ways relative to the task at hand. When the task at hand is the general nature of the domain in which the system operates, the ontology capture method will provide the appropriate perspective. The next tier in the vision is for all organizations — within the bounds of their proprietary interests — to have ontologies of their various component systems available for sharing and reuse. IDEF5 is being developed in the belief that it can contribute in a vital way to the realization of this vision of global knowledge sharing.

2 Conceptual Foundations of Ontology

The primary goal of the Ontology Description Capture method is to provide a structured technique, supported by automated tools, by which a domain expert can effectively develop and maintain usable, accurate, domain ontologies. In the IDEF5 method, an ontology is constructed by capturing the content of certain assertions about real-world objects, their properties, and their interrelationships and representing that content in an intuitive and natural form. This section provides an overview of the nature and content of an ontology, followed by a discussion that contrasts the ontology capture method presented in this report with other existing methods.

2.1 The Nature of Ontological Inquiry

Historically, ontologies arise out of the branch of philosophy known as *metaphysics*, which deals with the nature of reality, of what exists. The traditional goal of ontological inquiry in particular is to divide the world "at its joints," to discover those fundamental categories, or *kinds*, into which the world's objects naturally fall. So viewed, natural science provides an excellent example of ontological inquiry. For example, a goal of subatomic physics is to develop a taxonomy of the most basic kinds of objects that exist within the physical world (e.g., protons, electrons, muons). At the other end of the spectrum, astrophysics, among other goals, seeks to discover the range of objects that populate its domain (e.g., quasars, black holes, gravity waves). Similarly, the biological sciences seek to categorize and describe the various kinds of living organisms that populate the planet. Further examples of ontological inquiry can be observed in the fields of geology, psychology, chemistry, sociolinguistics, and, in general, any discipline that attempts to understand the nature of some set of physical, psychological, or social phenomena.

However, this sort of inquiry is not limited to the natural and social sciences. Abstract sciences as well — mathematics, in particular — attempt to discover and categorize the domain of abstract objects such as prime numbers, polynomial algorithms, commutative groups, topological spaces, and so forth.

The natural and abstract worlds of pure science do not exhaust the applicable domains of ontology. There are vast, human-designed and engineered systems such as manufacturing plants, businesses, military bases, and universities in which ontological inquiry is just as relevant and just as important. In these human created systems, ontological inquiry is primarily motivated by the need to understand, design, engineer, and manage such systems effectively. This being the case, it is useful to adapt the traditional techniques of ontological inquiry in the natural sciences to these domains as well.

2.2 The Central Concepts of Ontology

2.2.1 Kinds

The construction of ontologies for human engineered systems is the focus of this report. In the context of such systems, the nature of ontological knowledge involves several modifications to the more traditional conception. The first of these modifications has to do with the notion of a kind. Historically, a kind is an objective category of objects that are bound together by a common *nature*, a set of properties shared by all and only the members of the kind. More exactly, on the traditional notion, for every kind **K** there is a set **N** consisting of properties that are individually necessary and jointly sufficient for being a **K**; that is, **x** is a **K** if and only if **x** has every property in **N**. Moreover, and significantly, these properties are traditionally held to be *essential* to their bearers; that is to say, they are properties that their bearers could not possibly lack. For instance, it is reasonable to say that the nature of gold is to have the particular atomic structure that it has: everything that has this property is gold, and nothing that lacks it is gold. Furthermore, in contrast to nonessential, or *accidental*, properties like *shape*, this property is essential to every instance of gold: no instance of gold could possibly lack it; otherwise, it would not be gold. On the traditional conception, then, to divide the world at its joints via an ontology is simply to identify the nature of each relevant kind in a given domain.

While there is an attempt to divide the world at its joints in the construction of enterprise ontologies, those divisions are not determined by the natures of things in the enterprise so much as the roles those things are to play in the enterprise from some perspective or other. Because those roles might be filled in any of a number of ways by objects that differ in various ways, and because legitimate perspectives on a domain can vary widely, it is too restrictive to require that the instances of each identifiable kind in an enterprise share a common nature, let alone that the properties constituting that nature be essential to their bearers. Consequently, enterprise ontologies require a more flexible notion of kind. Toward this end, the first modification to be made is terminological. To avoid overloading the term "nature," call the set of properties associated with membership in a given enterprise kind **K** its *defining properties*. (Note: as with natures, defining properties are not properties exemplified by the kind K itself; they are properties exemplified by its *instances*, the **K**'s.)

Second, unlike the properties that make up a kind's nature in the traditional conception, the defining properties of a kind need not be essential to its instances. Depending on the kind in question, a defining property, this may or may not be so. Mathematical kinds provide the easiest examples of the former. The property **having four sides**, for example, can be taken as a

defining property of the kind **rectangle** (not the only one, obviously), and is also an essential property of all instances of the kind; no rectangle could possible fail to have four sides. On the other hand, because of its role in a certain manufacturing cell, the property **having a diamond insert** might be a defining property of the kind **cutter** in a certain enterprise, even though the particular cutter chosen to fill that role could, if desired, be fitted with a carbide insert instead. In this case, the defining property of the kind is accidental to (at least one of[1]) its instances. A property's status as a defining property, relative to a given kind, is thus independent of its being essential or accidental to instances of the kind.

By the same token, a property's status as essential to some or all instances of a kind is independent of its status as a defining property. That is to say, the two classification dimensions of *defining* and *essential* are orthogonal. For example, suppose the defining property of a circle is taken to be **being a closed planar figure all of whose points are equidistant from a given point**. Then, while not a defining property of the kind **circle**, the property **having no interior angles** is nonetheless essential to all of its instances — no circle could possibly have an interior angle. Finally, there may be properties of instances which are neither defining nor essential: for example, the number of pages of a given requirements document. These different possibilities are represented in tabular form in Figure 2-1.

	Defining	**Nondefining**
Essential	Kind: **Rectangle** Property: *having four sides*	Kind: **Circle** Property: *having no interior angles*
Accidental	Kind: **Cutter** Property: *having a diamond insert*	Kind: **Req'ts document** Property: *being 10 pages in length*

Figure 2-1. Defining/Nondefining vs. Essential/Accidental Properties

One final modification to the traditional notion of a kind remains. Instead of requiring that the defining properties of **K** are individually necessary and jointly sufficient for membership in **K**, it is

[1]This is yet another possible subtlety; properties essential to some members of a given kind may be accidental to other members of the kind.

required only that, for every instance **x** of a kind **K**, **x** have *at least one* of the defining properties of **K**. The reason for this weaker condition is that it often happens that, although instances may be easily recognizable, it may also be that there are no clear, exceptionless criteria for something's being an instance of a given kind; a domain expert might simply "know them when she sees them."[2] In such cases, one must rely upon the domain expert to stipulate when a given object is, in fact, an instance of the kind in question.

In a particular case, of course, when the defining properties of a kind fit the more traditional conception of a nature, the stronger conditions associated with natures can, and should, be added to the definition of the kind. The point of the weaker conditions is to allow something to count as a kind even without meeting the stronger conditions of the traditional conception. It is because of the weaker conditions on instancehood in our conception that we have avoided familiar related terms like "type" and "class" in favor of the less entrenched term "kind." In virtually all contexts, types and classes are taken to have unambiguous, well defined, necessary, and sufficient membership conditions; this is so, for example, for the notion of a type in Pascal or C, the notion of an abstract data type in the theory of programming languages, and the notion of class in the theory of sets. Though entirely appropriate in those contexts, such rigorously defined membership conditions are simply too inflexible to capture the subtleties of categorization and grouping in human engineered systems.

2.2.2 Kinds as Distinguished Properties

What then, exactly, are kinds? For the reasons just noted, they should not be identified with types or classes. Even if they are so identified, the question of what they are would not be settled until the same question is answered with regard to types and classes. What is distinctive of all three notions is the fact that they are what might be called *categorial*. Classes, types, and kinds all indicate some grouping of individuals into categories. Thus, all three are (typically) *multiply instantiable*; different individuals, that is, can be instances, or members, of the same type, class, or kind. Furthermore, in the case of types (in general) and kinds at least, these entities are *intensional*; that is to say, unlike sets (and perhaps classes, depending on the account), the identity

---

[2]The philosopher Wittgenstein illustrated this idea with the kind **Game**. Though games certainly appear to constitute a distinct kind of activity, there is no specific set of properties that are individually necessary and jointly sufficient for something to be a game. Some, but not all, games have rules; some involve scoring, while others do not; some are competitive; some have a time-limit; and so forth. Instead of a set of properties that *determine* whether something is an instance of the kind **Game**, there is a broad set of properties each of which is characteristic of some but perhaps not all instances of the kind. In any case, the properties provide nothing like a set of necessary and sufficient conditions for being a game.

of a type or kind — its being what it is — is not dependent upon its membership; the instances of a type or kind **K** can change over time without any change in **K** itself. The employees of an enterprise can grow, yet the kind **employee** — that in virtue of which a thing is rightfully considered an employee — persists; an aging aircraft can be replaced by a newer one without any effect on the kind **aircraft**, and so forth. These two characteristics — multiple instantiability and intensionality — however, are distinguishing features of what are typically called *properties*. Because properties are already a part of the basic metaphysics of IDEF5, it will, therefore, be both intuitive and convenient simply to take kinds to be properties of a certain distinguished sort.

2.2.3 Contrasting Properties and Attributes

It is important for the purposes of ontology that the terms "property" and "attribute" be clarified. An attribute is best thought of as a *function*, that is, a mapping that takes each member of a given set of individuals to a single specific value. Thus, the attribute **color-of** maps each object (in a given set) to its color; the attribute **age-of** maps each employee to his or her age. By contrast, a property is intuitively not such a mapping. Rather, they are just *characteristics* of things, "ways things are," abstract, general characteristics that individuals share in common.

Things exhibit certain attribute values: the **color-of** that object is red, hence, it has the property of **being red**; the **age-(in-years-)of** that employee is 40, hence, she has the property of **being 40 years old**. However, there is not always such a correlation between properties and attribute values to be found. For example, neither the property **having at least one interior angle** nor the property **having a color**, because of their indefiniteness, is obviously clearly correlated with any sort of attribute value. The usefulness of both properties and attributes in IDEF5 lies in precisely this observation. It is often the case that the defining properties of a kind will be *indefinite* with regard to any particular attribute value. For example, the property **has-identifiable-serial-number** might be a defining property of the kind **NC machine** in a given manufacturing domain. An NC machine's having this property, however, says nothing about what its *actual* serial number is. A modeler can, therefore, indicate that this is information that is to be kept about instances of the kind by including the corresponding attribute **serial-number-of** among the attributes associated with the kind in question.

Practice has confirmed that in the course of building an ontology it may initially be unclear whether an identified notion is best thought of as a property or as an attribute. Consequently, in the IDEF5 methodology, the term "characteristic" is used as a neutral term encompassing both properties and attributes (See Section 3).

2.2.4 Relations

So far, only properties and attributes of individuals have been considered. But, of course, there are other sorts of general features that individuals exhibit, albeit jointly rather than individually, namely, *connections*, or *associations*, or as they shall be referred to here, *relations*. The relation **works-in**, for instance, is a general feature that holds between an individual and the department in which he or she works. Like a property, then, it is both multiply instantiable — different pairs of things can stand in the same relation — and intensional — a relation's identity does not consist in its instances.

The relations in an ontology are *typically* binary; that is to say, they hold between two entities, as with the relation **works-in**. However, there is no theoretical bound on the "arity" (number of arguments) of a relation; the relation **between**, for instance, holds between three objects. More artificial but nonetheless useful relations can easily be defined with four or more arguments. IDEF5 thus places no restriction on the arity of the relations that can be introduced into an ontology.

2.2.5 Second-order Properties and Relations

Intuitively, properties and individuals are of different logical types. Properties are the abstract, general features that are *shared* by distinct individuals, the *respects* in virtue of which distinct individuals are the same. Similarly, relations are the general associations which can be shared by distinct pairs (triples, etc.) of individuals. Thus, properties and relations are identified by abstracting away from the particular features of individuals and, hence, are often characterized as being of a *higher* (i.e., roughly, more abstract) logical type than the individuals that exemplify them. Individuals are thus frequently referred to as *first-order objects*, and properties and relations of first-order objects as *first-order properties and relations*. However, properties and relations that hold among individuals are identifiable (albeit abstract) objects themselves. But because they are one level of abstraction above ordinary first-order objects, they are said to be of a higher *logical type* and, hence, classified as *second-order* objects. As objects, first-order properties and relations can themselves have properties that apply to them, but not to individuals: for example, the property **having at least one instance**. Such properties are typically known as *second-order properties*, because they apply to second-order objects. Furthermore, second-order objects can stand in relations with one another. The relation **has-more-instances-than**: for example, is a relation that holds between two kinds. Again, the **subkind** relation is a relation that holds between a given kind and a more general kind that subsumes it, for example, the kinds **human** and **mammal**, or **NC machine** and **machine**. Some second-order relations, however, include individuals among

their arguments. The **instance-of** relation, for example, holds between an individual **a** and a kind **K** just in case **a** is an instance of **K**. Such "mixed-type" relations that hold between objects of different logical types are nonetheless second-order. A second-order relation, therefore, is a relation that always includes at least one first-order property or relation among its arguments.

As with properties and relations holding of individuals, IDEF5 permits reference to any higher-order property or relation. The **subkind-of** relation and the **instance-of** relation, however, because of their prominence and importance, are included explicitly in the IDEF5 language. Note that both of the distinguished second-order relations — **subkind-of** and **instance-of** — are often ambiguously expressed by the expression "is-a" in semantic nets and other graphical languages. To avoid the possible confusions this practice might engender, they are explicitly distinguished in IDEF5, and the expression "is-a" is not used in the IDEF5 language.

2.2.5.1 More on the Subkind Relation

The subkind relation has an important consequence for individuals: if **K** is a subkind of **K′**, then every instance of **K** is an instance of **K′**. It should be noted that the converse does not hold in general: if every instance of a kind **K** is an instance of another **K′**, then it does not in general follow that **K** is a subkind of **K′**. The reason for this is that the subkind relation is *necessary*: it is not enough that, as a matter of fact, every **K** is a **K′**; it must also be *impossible* (in some appropriate sense) for a **K** to fail to be a **K′**. For instance, every **U.S.-president** is a **human-male**. But this is obviously a contingent relationship; there is nothing necessary about it, and indeed quite likely in the future there will be a woman president. By contrast, in the current U.S. government ontology (holding fixed certain basic laws about the presidency), **U.S.-president** is a subkind of **American-citizen**. The previous examples are more obvious still; clearly, no human being (as the notion is currently understood) could fail to be a mammal, just as no NC machine could fail to be a machine.

It should be noted also that the subkind relation encompasses several widely-used notions. The notion of kind subsumes those of type and class. A corollary to this is that the notion of subkind subsumes those of subtype and subclass. More exactly, types and classes are just kinds with definite necessary and sufficient membership conditions. A subtype **T** of a type **T′** is a type whose membership conditions entail those of **T′**. The same holds for classes. In such cases, it is also said that **T** is a subkind of **T′**, and such occurrences of the subkind relation are known as *description subsumption*, because the membership conditions of **T′**, the description of what it is to be a member of **T′**, subsume those of **T**.

By contrast, in those cases where a kind does not have rigorously specified necessary and sufficient membership conditions, the subkind relation, in general, can only be stipulated, not inferred. That is, in cases where it is only possible to declare that a given individual is a member of a certain kind **K**, the information associated with **K**, in general, will not provide a means for determining whether another kind **K′** is a subkind of **K**. The reason for this is that, if membership is not neatly determined for a given kind **K**, there will be no guarantee that membership in another kind **K′**, in particular, will be sufficient for membership in **K** (though it *might* be in specific cases).

The notion of subkind also encompasses the notion of *generalization/specialization*, that is, occurrences of the subkind relation in which the subkind is naturally thought of as a special case of a more general concept. For instance, the kind **hex-headed-bolt** is naturally characterized as a specialization of the concept **fastener**.

2.2.6 Two Ways to Introduce Kinds into an Ontology

Kinds are introduced into an ontology either by definition or stipulation. The former case is applicable when it is possible to provide necessary and sufficient conditions for a thing's being an instance of a kind **K** in terms of objects, properties, and relations already assumed to exist in the domain at hand. In the case of definition, then, the whole of **K**'s logical nature is given in terms of antecedently given elements of the ontology. In the latter case, **K** is postulated to exist, but only a *partial* definition of its nature is provided. To illustrate, the kind **prime number** can be totally defined in terms of the kind **number**, the number one, and the relation **divisible-by**. By contrast, a kind like **book** in a library ontology cannot be defined completely in terms of other kinds, properties, and relations in the domain; it is not, for instance, *just* the collection of pages. However, it can (let us suppose) be *partially* defined, or *axiomatized*, in terms of other kinds in the domain, for instance, the kinds **author** and **publisher**. Properties and relations in general can be introduced into an ontology by definition or stipulation as well.

2.2.7 Parts, Wholes, and Complex Kinds

The examples above might suggest that individuals are considered logically simple in IDEF5. However, it is clear that individuals of most kinds (people, NC machines, etc.) can themselves be viewed as complex objects comprising many other objects of various kinds. In general, individuals are considered simple only insofar as their composite nature is irrelevant to the particular perspective from which they are being viewed. By the same token, from some perspectives, the composite nature of a certain kind of object may be highly relevant; one might, for example, wish to document not only the existence of the kind **engine** but also the fact that this

kind of object includes objects of other kinds (e.g., distributors, pistons, sparkplugs) as parts. One might then want to document the decomposition of those objects as well. IDEF5, therefore, includes among its primitives a basic **part-of** relation that holds between an individual and the more complex objects of which it is a part. It thus holds between, say, a given spark plug and the engine in which it has been installed. When the members of a kind are viewed as having parts in a given ontology, the kind is known as a *complex kind.*

The **part-of** relation is characterized in IDEF5 simply as a *weak partial ordering* on the domain of individuals. That is to say, it is entirely characterized formally by the two (higher-order) properties of reflexivity (every object is a part of itself) and transitivity (every part of a part of an object *a* is a part of *a*). Thus, for instance, a spark plug that is a part of a car's engine is also a part of the car. It should be noted, though, that IDEF5 does not assume the full theory of parts and wholes; in particular, it is not assumed that the domain of individuals is closed under the formation of complex objects. For example, it is not assumed that, for every two individuals **a** and **b**, there is a third individual that is the "sum" of **a** and **b** (though, of course, one is free to *postulate* such a principle as part of the ontology of a given domain).

2.2.8 Processes, States, and Process Kinds

An adequate characterization of the kinds that inhabit a given domain often cannot be divorced from the *processes* in which their instances are involved. Typically, processes involve two sorts of change: change in *kind* and change in *state*. In an incineration process, for example, there is a transformation of some quantity of wood into ashes and gas; the wood is destroyed and quantities of ash and gas result. By contrast, a process in which ice is melted simply involves a change in state of a given quantity of water from frozen to liquid; the water itself is not destroyed, but only altered in a nonessential way. This is in fact what generally distinguishes states from kinds: unlike kinds, states are usually *contingent* groupings within a domain. That is, the distinguishing feature of a state is usually a changeable, accidental feature of a thing: for example, a quantity of water's being *frozen*, or a car body being *painted.* Both sorts of change are accomodated in IDEF5.

Like individual objects, processes, too, cluster naturally into general categories. For instance, temporally distinct events in which a manufacturing process plan is generated from a given design are all instances of the general process of manufacturing process planning. Thus, general processes, like the kinds discussed in the previous paragraph, are multiply instantiable; distinct, individual events can be instances of the same general process. Furthermore, the identity of a general process is independent of its instances; it remains the same regardless of whether or how it is instantiated. Hence, also like kinds, a general process is intensional. Therefore, general

processes can be thought of as kinds no less than object kinds. Unlike the instances of individual kinds, however, processes are *things that happen*. Thus, not only do they "contain" other objects as parts, like the instances of complex kinds, they *occur* over an interval of time, and things are *true of* the objects in the process over at least some parts of that interval. It is this fact that often makes it relevant to refer to relevant processes in the characterization of a given kind.

Because of the importance process kinds can have in the definition of a domain ontology, IDEF5 permits one to refer to them no less than object kinds. However, there are two distinct contexts in which such reference can occur, and the information that is kept about a process kind will differ depending on the context. If a process kind **P** is referred to in the description of a transformation or transition involving two kinds of objects, then the "internal" character of **P** is described in accordance with the IDEF3 process description capture method. That is, **P** is described in terms of the object kinds it involves, their properties, and the relevant relations that hold between instances of those kinds when the process in question is instantiated. In particular, in such contexts, the usual sort of information kept about an object kind — its defining properties and so forth — is not kept about the process kind.

On the other hand, it may be important for understanding a domain not only to know how objects are involved in the internal structure of a process, but also — as with object kinds generally — how one kind of process relates logically to another kind of process, independent (in general) of the details of its internal structure. For instance, **manufacturing process planning** is a subkind of **planning**. In these cases, process kinds are characterized exactly like object kinds: defining properties are identified, and so on. Two distinct constructs will be provided in the IDEF5 graphical language corresponding to these two possible characterizations of process kinds.

2.3 Levels of Ontologies

Roughly speaking, enterprise ontologies can be categorized in terms of three levels of generality, as shown in Figure 2-2. These levels are useful when scoping an ontology-building effort as discussed in Section 3. At the highest level of generality are *domain ontologies*. A domain ontology classifies the most general information that characterizes an entire domain. For example, a domain ontology for semiconductor manufacturing would include general information about products, manufacturing techniques and tools, and so forth, applicable across the entire semiconductor manufacturing domain. (The notion of a domain ontology is somewhat flexible in the sense that it might be possible to abstract further across various specific domain ontologies to derive an even more general domain ontology; for instance, one might abstract from semiconductor

manufacturing, the automobile industry, and so forth, to derive an ontology that encompasses manufacturing generally.)

At a lesser level of generality are *practice ontologies*. A practice ontology is an extension of a domain ontology that includes the common features of similar sites in that domain. For example, a group of semiconductor manufacturing companies involved in the development of similar product lines might develop an ontology that characterizes the semiconductor domain from the perspective of the development of that product line.

Finally, at the lowest level of generality are *site-specific ontologies*. A site-specific ontology extends a practice ontology (hence, also a domain ontology) to include information about all relevant kinds of objects, properties, and relationships found within a specific site. Thus, for example, beginning with a portable practice ontology developed independently on the basis of similar sites, a specific semiconductor manufacturing plant in Silicon Valley might use IDEF5 to extend this ontology to create a site-specific ontology to describe its own facilities in detail.

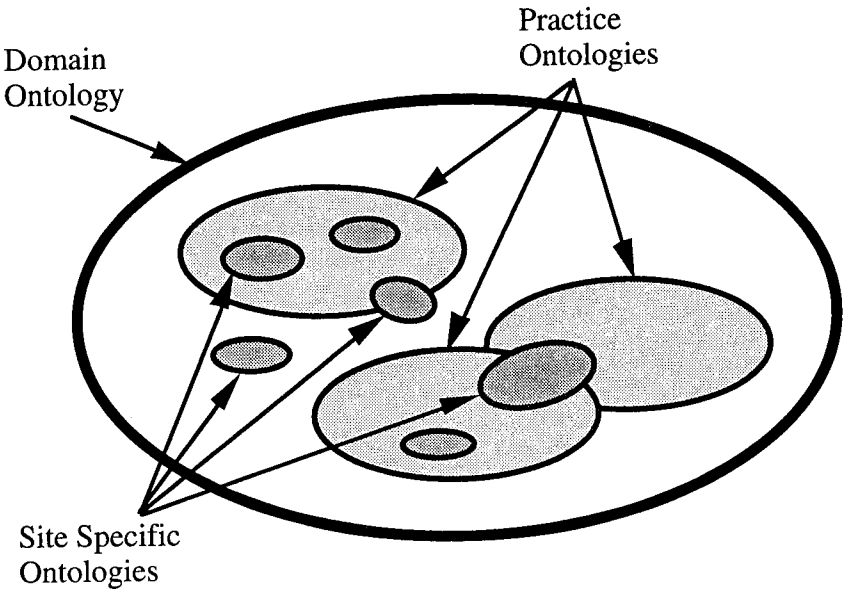


Figure 2-2. Levels of Ontologies

A traditional problem with ontology development has been that many practice and site-specific ontologies for the same sites may exist within a single domain [Hobbes 87] because what an ontology captures is influenced by the viewpoint of the developers. Recognizing this problem, IDEF5 facilitates the capture of ontologies from multiple viewpoints and suggests guidelines for resolving inconsistencies because of different frames of reference.

2.4 On the Need for a Separate Ontology Modeling Method

Ontology development has traditionally been a difficult and expensive task. Ontologies developed to date, such as Tacitus [Hobbes 87] and CYC [Lenat 86], are the result of very expensive and handcrafted efforts by highly skilled specialists. Many enterprises are unable to fund such expensive efforts. A standard and cost-effective means of developing ontologies must be developed if enterprises are to gain the benefits of ontology use. The development of a theoretically and empirically well-grounded *method* specifically designed to assist in creating, modifying, and maintaining ontologies will result in both standardized procedures and reduced costs.

The goal of IDEF5 is not to define yet another method to do something a little better or a little different than an existing method. Rather, the IDEF5 method development is designed to fill a gap in the existing set of methods. A type of information — ontological information — has not been directly targeted by any existing method. The importance of this sort of information should be clear. What is perhaps less clear is the need for a new method for capturing this information. This issue is taken up in this section.

For those familiar with other IDEF methods, the idea of capturing information about kinds and their associated properties will no doubt suggest both IDEF1 and IDEF1X. A kind has been defined in Subsection 2.2.1 as a certain sort of class, which might then suggest that a kind is like an IDEF1 *entity class* or an IDEF1X *entity*. Furthermore, associated with each entity class (entity) is a list of associated *attributes* that assign values to the members of the entity class. So perhaps the makings of an ontology modeling method are already available in one of these two methods.

However, this is not the case. The chief reason is that IDEF1 and IDEF1X are purposely designed with certain intrinsic expressive limitations to constrain the structure of the information that they represent. IDEF1X, for example, was designed explicitly for the design of relational database models; hence, it can only express information of a rather low logical complexity. This makes for very clear, uncluttered, and easily implemented data models, but it also limits the applicability of IDEF1 and IDEF1X outside their intended domains. For example, neither method has the capacity to express modal information. For instance, in the manufacturing cell example discussed earlier, it could be desired for security reasons that it be impossible to swap out the diamond tool in the cutter; that is, suppose it is specified that any instance of the kind **Cutter** *must* have a nonremovable diamond tool. Without the capacity to express modal information, it is not possible to represent this constraint. But as the example illustrates, it may be of singular importance to be able to express such information.

Further examples abound. For instance, in both IDEF1 and IDEF1X, it is not possible to name individual objects in an ontology and assert things specifically about them. Rather, one can only say things that hold of every member of a given class of entities in general. This is a crucial limitation in cases where there is a distinguished member of a given kind with special properties. Again, the two methodologies can express only a limited variety of general propositions about the structure of the entities within a given class. For instance, one might want to note that for every member of class A with property **P**, there is another member with property **Q**. This is a straightforward quantification statement, easily expressed in the language of predicate logic that underlies IDEF5; once again, though, this proposition is beyond the expressive capabilities of IDEF1 and IDEF1X. However, as with the previous examples, the capacity to express such information might well be needed in giving a thorough characterization of the nature of the objects within a system.

The main point here is that the existing IDEF methods were simply not designed to do ontology modeling; they were designed with other goals in mind. Granted, the methods could perhaps be extended to incorporate additional expressive power as the need arises, but there would be no point. A method designed for one purpose should not be forced to perform another. Again, the claim is not that there is something wrong with or inadequate about the existing IDEFs. They were simply not designed as methods for ontology development.

3 The IDEF5 Ontology Development Process

This section describes the IDEF5 ontology description development process. As described in Subsection 2.1, an ontology characterizes what exists: the kinds, their properties, and their interrelationships in a given domain, as revealed in the terminology used by experts in the domain. A complete ontology, then, reveals the fundamental nature of a given domain. In the context of business re-engineering, concurrent engineering, and information system development, an ontology is key for the design of effective scalable solutions.

Practically, an ontology is a documentation of the terminology used in a particular domain. It also includes the rules for combining and using that terminology to form statements about the domain, and sanctioned inferences that can be made from those statements in the domain. This use of ontology is consistent with the traditional use because what "exists" in a given domain is largely influenced by the *ability* of the agents to individuate or "carve up" the world. As humans differ greatly in this ability, both because of differing capabilities and differing perspectives, ontologies are rarely perspective-invariant. Ontology development is focused on understanding the concepts of a domain from these varied perspectives. It is also focused on extracting the essential nature of these concepts and representing this knowledge in a structured manner.

The construction of an ontology differs from traditional information capture activities in the *depth* and *breadth* of the information captured. Thus, an ontology development exercise will go beyond asserting the mere *existence* of relations in a domain; the relations are "axiomatized" within an ontology (i.e., the *behavior* of the relation is explicitly documented). To illustrate, consider the relation **higher than**. Typical information analysis activities (such as IDEF1X modeling) would assert the *existence* of this relation and declare its cardinality, only if the organization "managed information about" the kinds of objects in the domain that can participate in the relation. IDEF5 allows the characterization of the relation in greater detail. Thus, an IDEF5 model of the **higher than** relation might declare that this relation has the property of being *transitive*. Moreover, IDEF5 provides mechanisms for characterizing the nature of transitivity by means of appropriate *axioms* (i.e., rules and constraints governing the behavior of relations with that property). Axioms are recorded using the IDEF5 elaboration language (Section 4.2). These axioms may be used to make inferences (i.e., derive new knowledge from existing knowledge). The property of transitivity, for example, enables inferences of the following form: suppose it is given that **A** is higher than **B** and that **B** is higher than **C**. Then, by the transitivity of the **higher than** relation, it can be inferred that **A** is higher than **C**. Ontological analysis, therefore, facilitates the extraction of information that is conveyed, but not displayed, in an ontology.

The ontology development process requires considerable skill and experience primarily because of the following reasons:

- The knowledge about a domain is often very poorly documented, and exists primarily in the minds of a few domain experts. This domain knowledge is of two kinds: *procedural* and *declarative* [Musen 89]. Declarative knowledge is the type of knowledge that human experts often find easy to make explicit (i.e., humans are consciously aware of this type of knowledge). This knowledge can be inspected, abstracted, and applied in a variety of different contexts. On the other hand, it has been observed that domain experts find it difficult to make procedural knowledge explicit [Musen 89]. Research reveals that experts' awareness of what they know progressively degrades after repeatedly applying their know-how to specific tasks.[3] The nature of domain knowledge therefore makes the process of "extracting" knowledge from domain experts intrinsically difficult.

- Researchers have found that a person's prior knowledge of a domain area is critical for properly assimilating new information and clarifying areas of ambiguous interpretation. Therefore, knowledge engineers must make an effort to learn the domain expert's area to avoid the consequences of misunderstanding what the expert is trying to convey.

- The knowledge analysis needed for ontology capture requires considerable introspective thought. Typically, it requires the effort of a group working in close concert, with ontology evolution occurring iteratively by a process of successive refinement [Musen 89].

Ontology development requires extensive iterations, discussions, reviews, and introspection. Knowledge extraction is usually a discovery process and requires considerable introspection. It requires a process that incorporates both significant expert involvement as well as the dynamics of a group effort. Given the open-ended nature of ontology analyses, it is not prudent to adopt a

---

[3]The problem for knowledge engineers is that experts do not introspect reliably. Although human beings may have some declarative knowledge of the extent of their procedural memory, the two types of memory appear to be handled quite separately by the nervous system. For example, Cohen investigated patients with neurologic amnesia to learn more about the mechanisms of human memory. In one experiment, 12 such patients were taught how to solve the Tower of Hanoi puzzle. The patients with amnesia became proficient at the task just as quickly as did control subjects without amnesia and learned rapidly to perform the necessary sequences of moves "without thinking". However, despite their obvious acquired expertise at solving the Tower of Hanoi puzzle, not one of the amnesia patients would ever state that he was familiar with the puzzle or knew its solution!

"cookbook" approach to ontology development. We recommend the use of a general procedure along with a set of useful guidelines. This section describes the ontology development process for potential users of the IDEF5 method.

The IDEF5 ontology development process consists of the following five activities.

- **Organize and Define Project** This activity involves establishing the purpose, viewpoint, and context for the ontology development project and assigning roles to the team members.
- **Collect Data** This activity involves acquiring the raw data needed for ontology development.
- **Analyze Data** This activity involves analyzing the data to facilitate ontology extraction.
- **Develop Initial Ontology** This activity involves developing a preliminary ontology from the acquired data.
- **Refine and Validate Ontology** This activity involves refining and validating the ontology to complete the development process.

Although these activities are listed sequentially, there is a significant amount of overlap and iteration between the activities. Thus, the initial ontology development (Activity # 4) often requires the capture of additional data (Activity # 2) and further analysis (Activity # 3). Each of the five activities will involve other activities and tasks. The remainder of this section will describe the ontology development process in greater detail.

3.1 Organize and Define the Project

3.1.1 Organize the Project

An important initial step in developing an IDEF5 ontology description is the formation of a development team. Each member of the IDEF5 team will perform a particular role in the development effort. Individuals who are involved in the modeling may each fulfill several roles, but each role is dealt with distinctly and should be clearly separated in the minds of the participants. The following are the roles assumed by IDEF5 development project personnel.

- **Project Leader** This administrative role is responsible for overseeing and guiding the entire IDEF5 development effort. This person is ultimately responsible for the

outcome of the ontology development effort, team organization and leadership, and schedule and budget management.

- **Analyst/Knowledge Engineer** This technical role is filled by a person with IDEF5 expertise who will be the primary developer of the IDEF5 ontology description. The person filling this role may be a regular employee of the organization requiring the ontology development, or the person may be hired on contract for the task. In the latter case, the organization requesting the ontology development is referred to as the Client.

- **Domain Expert** This role is played by the primary source of knowledge from the application domain of interest. Persons filling this role will provide insights about the characteristics of the application domain that are needed for extracting the underlying ontology knowledge. Often the knowledge of a domain expert is supplemented by a variety of different documents from the organization. The supporting documents are referred to as source material.

- **Team Members** All persons involved with the IDEF5 ontology description project.

- **Reviewers** Persons knowledgeable about the application domain and/or the IDEF5 method responsible for reviewing and commenting on draft descriptions and documents. Reviewers authorized to make written critiques of IDEF5 descriptions are commentors. The remaining reviewers are called readers. Team members and domain experts can be reviewers.

3.1.2 Define the Project

The development team must establish the purpose and context of the description capture effort as early as possible in the project. The context statement bounds or delimits the area of the domain addressed by the development effort. The context is established by scope statements and the identification of the initial boundaries for the ontology acquisition project. The scope defines the boundaries of the description development effort and specifies parts of the systems that must be included or must be excluded. The purpose statement provides a set of "completion criteria" for the ontology description capture effort. The purpose is usually established by a list of 1) statements of objectives for the effort, 2) statements of needs that the description must satisfy, and 3) questions that must be answered by the resulting ontology description. The purpose and context can rarely be determined completely and accurately in advance. The list of needed findings or questions (the purpose) should be periodically revised as the data starts being compiled. Similarly,

the context an analyst thinks will contain the data often turns up leads to other areas not originally considered within the scope. Thus, the purpose and context generally evolve and are refined throughout the duration of the project. The purpose, context, and viewpoint of an IDEF5 ontology description are captured in an IDEF5 Description Summary Form, as shown in Figure 3-1. The IDEF5 Description Summary Form also references the document numbers of all the different artifacts that comprise the IDEF5 description.

3.1.2.1 Define Purpose

The statement of purpose clearly specifies the main objective(s) of the ontology development effort. Defining the purpose is an important initial step in the development effort. Often, project personnel take the purpose for granted only to find the results of their efforts ignored by or of little use to the enterprise. Without a purpose statement, the only completion criteria is the budget and time allocated to the effort. Conversely, with a clearly defined purpose, the project can often be completed much more cost effectively. Defining the purpose can be separated into two parts: 1) defining a statement of need (SON) and 2) defining objectives of acquiring and maintaining the ontology.

The SON should identify the source of the request (person or project) and paraphrase the stated motivations for the project. Identifying the objectives is simplified by answering the following questions.

- What decisions must be supported by the ontology description?
- How much detail is needed in the ontology to resolve an issue, make a decision, or answer a question?
- What question(s) does the client or domain expert need answered?
- Who will use the ontology once it is available?

3.1.2.2 Determine Scope and Level of Detail

Once the purpose of the effort has been characterized, it is possible to define the context of the project in terms of 1) the scope of coverage and 2) the level of detail for the ontology development effort. The scope defines the boundaries of the description development effort, and specifies parts of the systems that must be included or excluded. A sample context statement from the semiconductor domain might be: "This is a site-specific description of the concepts and terminology associated with the shop floor level objects in the domain" (e.g., wafers, etchants,

wafer carriers, automated ground vehicles). This context statement indicates that the description will not cover scheduling, manufacturing cells, bills of materials, or many other possible aspects of the semiconductor manufacturing domain. The level of detail specification is normally documented in the form of a set of examples. It should be noted, however, that the scope and level of detail decisions are tentative at this stage of the project and should be updated as the ontology data becomes available. An astute project leader will periodically assess the adequacy of the ontology captured against the specified needs and information goals of the client.

3.1.2.3 Establish Viewpoints

Although ontology is the study of the ". . . nature of being, reality, or ultimate substance," [Webster 88], ontologies are tinted, just as any other data/information/knowledge, through the eyes of the individual responsible for recording the description. Different individuals perceive the world around them in (often significantly) different ways because of differences in cognitive skills and background knowledge. These differences in humans' *capacity to individuate* are reflected in the *perspective* or *viewpoint* that every individual brings to bear on everyday activities. Differing viewpoints therefore have a significant impact on the outcome of an ontology capture effort.

The differences in viewpoints, or frames of reference, are often reflected in different aspects of the ontology such as the definition of the scope of the entire effort, the definition of boundaries between subsystems, and the specification of the level of detail of the description capture. A sample viewpoint statement for an automobile manufacturing ontology might be "described from the viewpoint of the production engineering department." Although the focus domain can be studied from different viewpoints, each IDEF5 description requires the selection of a specific viewpoint.

An IDEF5 description can have a set of different viewpoints associated with it. Resolving differences in viewpoints is an important part of ontology analysis. The viewpoints must be explicitly recorded in the IDEF5 Ontology Description Summary Form (Figure 3-1).

3.1.2.4 IDEF5 Description Summary Form

The IDEF5 Description Summary Form summarizes the evolving/completed ontology description. It records the purpose, viewpoint, and context and also provides a summary of all the schematics and documents used to record the ontology. The following are the fields of an IDEF5 Description Summary Form (see Figure 3-1).

IDEF5 Description Summary Form		
Project: Project Planning Ontology	**Analyst:** P. Benjamin	**Reviewer:** R. Mayer
Version: 1.0	**Review Starting Date:** 5/10/94	**Review completion Date:** 10/15/94
Purpose: To develop an ontology of the project planning domain.		
Context: The information acquired must be sufficient to plan activities, specify precedence relationships, and assign resources to activities.		
Viewpoint: Project Manager		

List of Documents

Source Material Log	Kind Pool
Source Statement Pool	Kind Specification Form
Source Statement Description Form	Property Pool
Term Pool	Attribute Pool
Term Description Form	Relation Pool
Proto-Kind Pool	Relation Specification Form
Proto-Kind Specification Form	Classification Schematic
Proto-Relation Pool	Composition Schematic
Proto-Relation Specification Form	Relation Schematic
Proto-Characteristic Pool	

Figure 3-1. IDEF5 Description Summary Form

- **Project Name** The name of the ontology description development project is recorded in this field. This purpose of this field is to identify the domain for the ontology description capture effort.

- **Version** This field records the version number of the ontology description. The version number is important because it provides a means to document and trace the evolution process of the ontology development.

- **Analyst** This field records the signature of the IDEF5 expert who is the primary developer of the IDEF5 description. It is important to record the name of the analyst responsible for the ontology development because the domain ontology reflects his/her viewpoint, individuation schemes (ways of "carving up" domain), and analytical skills.

- **Review Start** This field captures the date of dispatch of ontology for review.

- **Reviewer** The signature of the reviewer is recorded in this field. This information is useful for future reference.

- **Review Completion** This field captures the date of review completion. The difference between the review completion date and review starting date indicates how much time a reviewer has taken to provide recommendations, insights, and comments on the ontology development project.

- **Purpose** The purpose of the ontology description development project is recorded in this field. The purpose of the domain ontology development is important because it provides a brief and concise description of what to expect from this ontology document.

- **Context** The context of the ontology description development project is recorded in this field. The context comprises the boundaries and the level of detail. The statement of context is important because it indicates the scope and level of granularity of the study.

- **Viewpoint** This field records the viewpoint of the ontology development project. Knowledge of the viewpoint provides clues about the rationale for the schemes of individuation used to carve up the domain.

- **List of Documents** The name(s) of the IDEF5 document(s) are recorded on an IDEF5 Description Summary Form. This information is important in order to have a basic idea of how anontology is developed and organized in each document, and serves as an index for each document.

3.2 Collect Data

The definition of viewpoint, context, and purpose sets the stage for the data gathering phase of the ontology capture effort. One problem with data collection is determining the appropriate sources of data. Experience indicates that the main data sources are the domain expert and documents relevant to the circumscribed ontology. It may be also instructive to scrutinize existing and relevant IDEF models in the organization. IDEFØ models and IDEF3 descriptions are likely to be sources of data.

The knowledge engineer/analyst must work closely with the domain expert to effectively record all the data relevant to the description development effort. The data collection process is both iterative and interactive. The process is iterative because the result of compiling/organizing the data will

drive additional data acquisition efforts. The interactions are necessary to make clarification and discover insights based on discussions with the domain expert. The data collection may occur in different modes: 1) direct transcription of data from source documents, 2) interviews and protocol analysis with domain experts, or 3) introspected observation of particular organization activities/phenomena.

Direct Questioning (Interview) and Protocol Analysis are the most commonly used knowledge-elicitation methods for acquiring knowledge from domain experts. An expert's response to a question may depend on the type of questions asked by a knowledge engineer during an interview [Musen 89]. The typical questions asked during an interview are described in Subsection 3.2.1.2.

3.2.1 Interview Guidelines

3.2.1.1 Interview Preparation

Data that needs to be acquired directly from a domain expert often will be obtained through interviews. The following general guidelines are suggested to prepare for the interview.

- Obtain background information about each expert who will provide potentially useful information including information about the responsibilities, current assignments, and other areas within or related to the domain in which the expert has experience. The name, location, telephone number, and e-mail address of the expert(s) should also be recorded.

- Prepare a brief outline of: 1) the purpose of the interview with the expert, 2) the topics to be covered, 3) the types of information being sought, 4) the authority for requesting the interview, and 5) the relevant questions that can be used to motivate discussions.

- Schedule a date and time for the interview with the expert.

3.2.1.2 The Interview

The interview with the expert is critical. The knowledge engineer/analyst (interviewer) should create a positive, friendly atmosphere during the interview. The interviewer should attempt to convey to the domain expert the feeling that they are working together to create the required ontology and solve some problem for the organization. Novice interviewers should constantly remind themselves that the experts are the ones with the knowledge about the domain. Generally, experts are interested in helping and often provide questions and lines of investigation that the interviewers had not thought of pursuing. Well-prepared interviewers will find that experts

provide far more information than was expected, often covering topics the interviewer had not anticipated. In an ontology description capture project, this is the bonus for good preparation.

The guidelines below should be considered when preparing questions for an interview.

- Questions should not interfere with the domain expert's line of thought. Research has shown that very *detailed* questions are often counter-productive. Examples of such questions are, "What RPM should a machinist maintain while performing a drilling operation?" or "Why didn't you consider performing Operation X before Operation Y?"

- Questions should prompt experts to express their thought processes during problem solving. When domain experts remain silent for a considerable amount of time, it is possible that they are solving the problem in their minds and not expressing all their thoughts. Prompting questions are useful at this stage. Examples of *prompting* questions are, "At this point of time, what are you thinking?," "What are you considering now?," and "What are your reasons for doing this?"

- Questions should cover details regarding not only frequently occurring situations but also rare situations in the domain.

3.2.2 Protocol Analysis

A protocol is an underlying pattern or structure of a discourse or behavioral process. The term protocol implies that an expert is solving a problem using commonly used approaches and tools. Protocol analysis is the process of analyzing a record of discourse or behavioral process. There are two types of protocol analysis: verbal protocol analysis and movement protocol analysis [Jackson 90]. In verbal protocol analysis, experts are asked to think aloud as they are solving problems. Knowledge engineers record the entire discussion during the problem-solving process. In movement protocol analysis, industrial engineers identify idle movements by studying motion efficiency.

3.2.3 Data Collection Documents

Regardless of the data collection methods used, it is important to establish an action plan for collecting data pertinent to the purpose and viewpoint of the model. Once collected, each piece of collected data must be traceable back to its source. Traceability of source material is important because it is the data that provides objective evidence for the basic ontology structures that are later isolated from this data. We suggest the use of six important support documents to facilitate source

data traceability: 1) Source Material Log, 2) Source Material Description Form, 3) Source Statement Pool, 4) Source Statement Description Form, 5) Term Pool, and 6) Term Description Form. These documents are described in greater detail later in this section.

3.2.3.1 Source Material Log

The Source Material Log is a document that serves as the primary index to all source material collected and utilized in the project. Each piece of source material is sequentially assigned a unique identifying number as the log is filled out (Figure 3-2). A source material may be a text book, a research article, an enterprise-specific document such as a policy manual or a procedure manual, a set of an interview notes, or direct observation notes. A Source Material Description Form (Figure 3-3) is filled out to describe each source material in greater detail. The following fields are used in the Source Material Log (see Figure 3-2):

Source Material Log

Project: Project Planning Ontology — **Analyst:** P. Benjamin

Source Material #	Source Material Name	Collected From	Collected By	Date of Collection
SM #1	"Production and Operations Analysis", by Nahmias, S., *Richard D. Irwin Inc.*, 1989	----	Hari	5/20/93
SM #2	Interview Notes	Phillips	Hari	5/25/93

Figure 3-2 Source Material Log

- **Source Material #** The reference number assigned to each source material is documented in this field. It is important to maintain source material numbers to provide traceability to the source material from which statements, terms, kinds, properties, etc. are individuated by member(s) of the ontology development team.

- **Source Material Name** The name of the source material is recorded in this field. The name of the source material provides a basic idea of whether it is a textbook, a research article(s), an enterprise-specific document such as a policy manual(s), procedure manual(s) or so forth, interview notes, or direct observation notes.

- **Collected From** The name of the person from whom the source material was collected is recorded in this field. If the source material is a set of interview notes, the name of the interviewee will be recorded in this field. This information is important for traceability, especially when many domain experts are interviewed during data collection. On the other hand, if the source material is something like a text book, a research article, etc., then this field is empty.

- **Collected By** The name of the individual(s) who collects the source material is documented in this field. It is important to document information about who is responsible for the collection of each source material for future reference.

- **Date of Collection** This field represents the date(s) on which the source material was/were obtained.

3.2.3.2 Source Material Description Form

The Source Material Description Form provides a summary of the source material information. For each source material item referenced in this log, there is a Source Material Description that is used to record more detailed information. The following fields are used in a Source Material Description Form (see Figure 3-3).

Source Material Description Form	
Project: Ontology of Project Planning	**Analyst:** P. Benjamin
Source Material #: SM #1	
Source Material Name: "Production and Operations Analysis," by Nahmias, S., *Richard D. Irwin Inc.*, 1989. Chapter # 8 (Project Scheduling)	
Purpose: To record the relevant source statements that help individuate ontology elements in the project planning domain.	
Comments: This source material concerns production in the broadest sense of the word: that is, production of goods and services.	
Abstract: "Project Scheduling" chapter focused on: 1) the network representations of a project, 2) the Critical Path Method (CPM), 3) project costing, 4) the program evaluation review technique method (PERT). 1) Networks are a convenient means of representing a project. There are two ways of using networks to represent projects: activity-on-arrow and activity-on-node. 2) The critical path is the longest path or chain through the network. The length of the critical path is the minimum project completion time, and the activities that lie along the critical path are known as critical activities. Delay in critical activity delays the project. This chapter presents a method, involving forward and backward passes through the network, that specifies the earliest and latest starting and ending times for all activities. 3) The goal of the project costing analysis is to determine the optimal time to perform the project that minimizes the sum of indirect and direct costs. Direct costs include labor, material, and equipment. Indirect costs include costs of rents, interest, and utilities. 4) PERT is an extension of critical path analysis to incorporate uncertainty in the activity estimates.	
Terms Supported: T #1, T #3, T #4, T #5, T #6, T #7, T #9, T #10	
Statements Supported: SS #1, SS #3, SS #10, SS #12, SS #13, SS #15	

Figure 3-3 Source Material Description Form

- **Source Material #** The source material number is recorded in this field. The source material number is important because it provides traceability to the source material from which statements, terms, kinds, and properties are individuated by member(s) of the ontology development team.

- **Source Material Name** In this field, the name of the source material is documented.

- **Purpose** The reason(s) for acquiring the source material are recorded in this field because future readers of the ontology document might be interested in knowing the purpose of collecting source material in the first place.

- **Comments** Any additional relevant remarks that help describe or justify the collection of source material are recorded in this field. This field is important because it allows a team member(s) to record special features or comments that are worth referencing at a later date.

- **Abstract** A summarized description of the source material is documented in this field. The abstract is important because it provides a concise overview of the main concepts discussed in the source material.

- **Terms Supported** The list of term numbers supported by the source material is documented in this field (see Subsection 3.2.3.5). The terms supported are important because they provide traceability for all the terms that are identified based on this specific source material.

- **Statements Supported** This field represents a list of statement numbers supported by the source material. It is an important field because it provides traceability to all statements that are identified based on a specific source material.

The Source Material Log, together with the Source Material Description, establishes important links between the ontology and the source knowledge that the ontology embodies. The information contained in the source material is the basis for much ontology discovery that occurs during ontology development.

3.2.3.3 Source Statement Pool

The Source Statement Pool records meaningful statements made by different individuals, as well as statements extracted from source documents during the ontology development effort. Each source statement is given a unique identification number to improve traceability. The following fields are used in a Source Statement Pool (see Figure 3-4).

Source Statement Pool		
Project: Project Planning Ontology		**Analyst:** P. Benjamin
Source Statement #	**Source Statement**	**Supported by**
SS #1	Project definition is comprised of project statement, project goals, personnel, and resources required for the project.	Caraway, Lingineni, & Hari
SS #2	Resources may be classified as personnel, computer systems, and facilities.	Caraway
SS #3	The project planner determines the sequence of activities to be performed, and specifies precedence constraints to be maintained.	Hari

Figure 3-4. Source Statement Pool

- **Source Statement #** The unique identifier assigned to a statement is recorded in this field. It is very important to maintain this information because the source statement numbers provide traceability to the evidence from which ontology elements such as proto-kinds, proto-relations, proto-properties, etc. are individuated.

- **Source Statement** The source statement itself is recorded in this field. This field will contain the latest version of the source statement. Older versions of the source statement can be obtained from the source statement description form (Figure 3-5).

- **Supported by** The name(s) of the team member(s) responsible for the identification of the source statement are recorded in this field.

The Source Statement Pool together with the Source Statement Description Form(s) allows ontology development team member(s) to document the evolution process of source statement(s).

3.2.3.4 Source Statement Description Form

The following fields are used in a Source Statement Description Form (Figure 3-5).

Source Statement Description Form		
Project: Project Planning Ontology	**Analyst:** P. Benjamin	
Source Statement #: SS #1	**Statement #S Evolved To:**	**Status:** *Active / Retired*
Source Material #: SM #1	**Statement #S Derived From:**	*Original / Derived*
Source Statement: Project definition is a clear statement of the project, the goals of the project, and the resources and personnel that the project requires.		**Supported by:** Caraway, Hari
Version 1: Project definition comprises the project statement, project goals, and resources (including personnel) required for the project.		**Supported by:** Caraway, Hari, Lingineni
Version 2:		**Supported by:**
Version 3:		**Supported by:**
Comments:		

Figure 3-5. Source Statement Description Form

- **Source Statement #** The unique identifier assigned to a source statement is recorded in this field. It is very important to maintain this information because the statement number provides traceability to the evidence from which ontology elements such as proto-kinds, proto-relations, proto-properties, etc. are individuated.

- **Source Material #** This field represents the number(s) of the source material items that support a source statement. When it is important to know the supporting source material from which the source statement(s) is identified, the source material number(s) provides a pointer to the source material.

- **Status** The statement is categorized as either *active* or *retired* and as either *original* or *derived*, and this categorization is recorded in this field. When a source statement is (still) used for data analysis purposes, the status of the statement is active. If a source statement is no longer used for data analysis purpose(s), the status of the source statement is retired. When a statement is directly collected from a source material, the status of that statement is original. On the other hand, if a statement is derived from a

set of statements, the status of that statement is derived. The status of a source statement provides an indication of where the statement is located in its evolution process.

- **Statement # evolved to** The source statement numbers that were generated using this statement as the base will be recorded in this field.

- **Statement #s derived from** The source statement numbers that were used to derive this source statement will be recorded in this field.

- **Source Statement** This field provides space to document the statement itself. It is important to document the source statement because it provides the necessary evidence based on which various ontology elements are individuated.

- **Supported By** This field represents the name of the team member(s) responsible for the identification of the source statement from a source material.

- **Version #** A source statement might undergo many successive refinements during the ontology development process. This field records the version number of the statement, which is incremented in chronological order. The versions of the statement are useful for recording the evolution of the thought processes that underlie the data capture and for helping in the ontology extraction process.

- **Comments** This field documents any additional information about the statement for future reference.

3.2.3.5 Term Pool

The meaningful terms relevant to the ontology development project effort are recorded alphabetically in a Term Pool. Terms often evolve to proto-kinds, proto-characteristics, proto-relations, kinds, characteristics, and relations (Figure 3-6). A term that is identified by a member(s) of an ontology development effort may be promoted to a proto-kind, proto-relation, or a proto-characteristic at a later stage. The following fields are used in a Term Pool (see Figure 3-6).

Term Pool				
Project: Project Planning Ontology		**Analyst:** P. Benjamin		
Term #	**Term**	**Source Statement Reference**	**Source Material Reference**	**Support List**
T #11	Project	SS #1	SM #1	PK #1
T #12	Project goal	SS #1	SM #1	PK #2
T #13	Resource	SS #1	SM #1	PK #3, PR #5

Figure 3-6. Term Pool

- **Term #** The unique identification number assigned to a term is documented in this field.
- **Term** This field records the term itself.
- **Source Statement Reference** The source statement(s) based on which the term is individuated by a team member is documented in this field.
- **Source Material Reference** The unique identification number of source material that is used to individuate a term is recorded in this field. This information is very useful for determining which statement is the basis for individuating a term and also which source material is used for collecting that particular statement.
- **Support List** The list of proto-kinds, proto-characteristics, and proto-relations that are supported by the term are documented in this field.

The Term Pool provides a list of the terms used to derive the ontology. Each term in the Term Pool is described in greater detail using the Term Description Form (see Figure 3-7).

Term Description Form		
Project: Project Planning Ontology		**Analyst:** P. Benjamin
Term #	**Term**	**Description**
T #3	Event	An event is a characterized point in time that has some significance to a real-world process.
T #10	Process	A process is a collection of interrelated activities that produces a set of outputs from a set of inputs.
T #12	Resource	Resources are objects/personnel that are consumed, used, or required to perform activities and tasks. Resources play an enabling role in processes.

Figure 3-7. Term Description Form

The following fields are used in the Term Description Form (see Figure 3-7):

- **Term #** Each term individuated by an analyst from a statement is given a unique identifier for future traceability, and the unique number is recorded in this field.
- **Term** This field documents the term itself. This term may be promoted as a proto-kind, proto-characteristic, or a proto-relation during data analysis.
- **Description** A description of the term is recorded in this field. Often, it is useful to document a concise description of the term during the ontology development process.

3.3 Analyze Data

The objective of this task is to analyze the source material to construct an initial ("first pass") characterization of the ontology. This task is performed by a team consisting of knowledge engineer(s)/analyst(s) and domain expert(s). This task will typically involve the following activities.

- List the objects[4] of interest in the domain. Some objects will be fairly obvious from an initial study of the source data (such objects are called phenomenological naive

[4]The term object is used in a generic sense to denote instances and kinds and also physical things and conceptual things.

objects). For example, the different kinds of machines that are on the shop floor will be obvious ontology candidates for a manufacturing system ontology. The level of detail that needs to be employed to develop this list will be guided by the viewpoint and context statements constructed earlier in the development process.

- Identify objects that are on the boundaries of the ontology. The initial boundaries defined in the context statement may need to be re-drawn to facilitate better conceptual structuring of the ontology. For example, recognizing that an Autonomous Guided Vehicle (AGV) is on the (initial) boundary between the machine cell subsystem and the stores subsystem may cause the boundaries to be re-drawn to include material handling equipment, and, hence, the AGV.

- Look for and individuate internal systems within the boundary of the description. Systems are defined as collections of physical and/or conceptual objects that work together to achieve common objective(s). Organizing ontologies by the system in the domain provides a clear conceptual framework for subsequent analysis of ontology knowledge.

The activities involved in the analysis of source material are summarized in Figure 3-8.

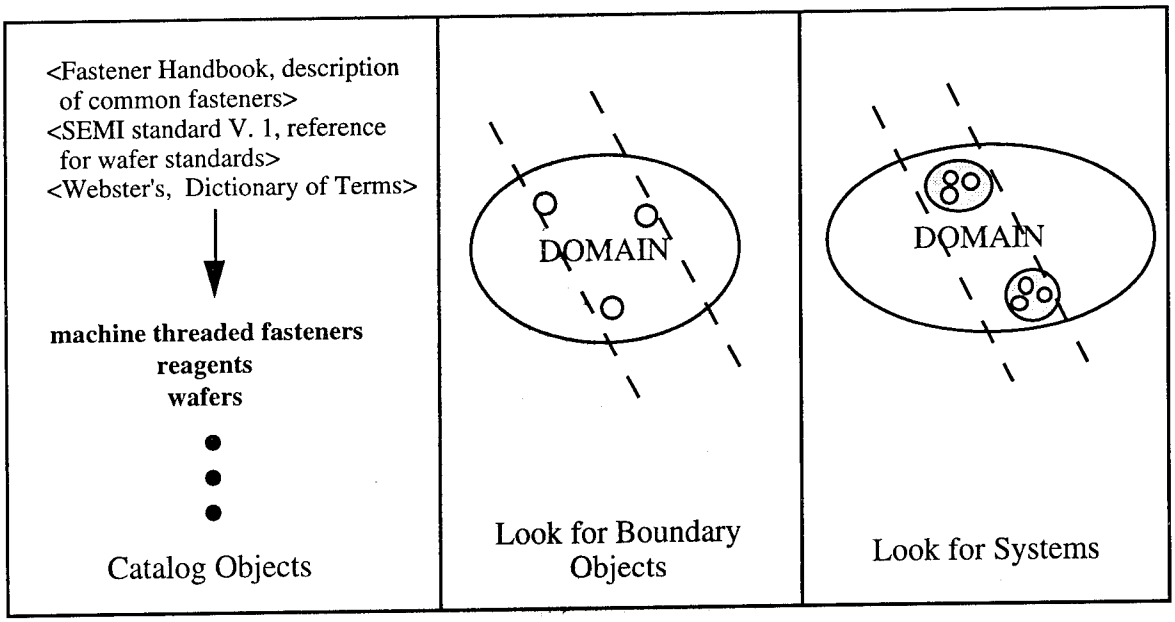


Figure 3-8. Source Material Analysis

3.4 Develop Initial Ontology

3.4.1 Develop Proto-Concepts

The term *proto-concepts*, in the IDEF5 method, refers to the set of proto-kinds, proto-properties, proto-attributes, and proto-relations. The prefix *proto-* suggests that the concepts are tentative and subject to further inquiry before final change of status. During the process of ontology development, proto-concepts often mature into concepts (i.e., kinds, properties, attributes, and relations). The kind refinement procedure and relation refinement procedure are described in Subsection 3.5.1 and Subsection 3.5.2, respectively.

3.4.2 Develop Proto-Kinds

The objective of this task is to convert one or more objects that result from the source data analysis to proto-kinds (if this is appropriate). A proto-kind is the result of a preliminary attempt at individuating a kind. This task essentially consists of associating the objects identified in Task 1 (Subsection 3.1) with the proto-properties identified in Subsection 3.4.3. It may be instructive to perform this association process in two stages. First, the association is carried up to the point where the proto-kind can be clearly distinguished from any other proto-kind; that is, the proto-kinds have a basis for being *uniquely* individuated. These properties that contribute to the uniqueness of a kind are candidate *defining properties*. Defining properties stipulate necessary conditions for membership to a kind. Once the defining properties are identified, other (non-defining) properties and attributes are used to characterize the kinds in greater detail. The proto-kinds are tagged with supporting source material (for traceability) and catalogued in the Proto-Kind Pool[5] (Figure 3-9).

---

[5]When a proto-kind matures into a kind during an ontology development process, the kind is recorded in a kind pool. The design of a kind pool is same as that of the Proto-Kind Pool, as shown in Figure 3-10.

Proto-Kind Pool					
Project: Ontology of Process Planning			**Analyst:** P. Benjamin		
Proto-Kind #	**Proto-Kind Name**	**Supported By**	**Supported By**	**Supports List**	**Schematic List**
PK #1	p[6]- Activity	Caraway	SM #1	PR #11	COS #1
PK #2	p- Resource	Caraway	SM #1	PR #12	CLS #2
PK #3	p- Project Plan	Caraway	SM #1	PR #13	RLS #1

Figure 3-9. Proto-Kind Pool

The various fields in a Proto-Kind Pool Form are briefly described below.

- **Proto-Kind #** The unique identifier of a proto-kind is recorded in this field. This proto-kind number is important because it provides traceability to the proto-kind.
- **Proto-Kind Name** The name of the proto-kind is recorded in this field. This proto-kind may be promoted to a kind at a later stage by team members during the ontology development process.
- **Supported By** The source material items (terms, statements) that support the process of individuating the proto-kind are documented in this field.
- **Supports List** The properties and relations that are identified based on the individuation of this proto-kind are recorded in this field.
- **Schematic List** The schematics in which this proto-kind occurs are documented in this field. Thus this field is important because it acts as a pointer to all the schematics in which the proto-kind is an element.

Each proto-kind has a Proto-Kind Specification Form (Figure 3-10) in which a brief description of the proto-kind, its synonyms, and other relevant comments can be documented.

---

[6]Prefix "p" denotes the notion of "proto" (i.e., proto-characteristic, or proto-kind, or proto-relation).

Proto-Kind Specification Form	
Project: Ontology of Project Planning	**Analyst:** P. Benjamin
Proto-Kind #: PK #1	
Proto-Kind Name: Resource	
Description: Resources are individuals that are consumed, used, or required to perform activities and tasks. Resources play an important role in processes.	
Synonyms: Machines, Equipment, Personnel, etc.	
Comments: Individuated by B. Caraway	

Figure 3-10. Proto-Kind Specification Form

3.4.3 Identify Proto-Characteristics

The objective of this task is to catalog the characteristics (that is, the properties and/or attributes) needed to identify and describe the objects in the domain. Properties and attributes are the characteristics that hold of objects in the real world. Examples of attributes are weight, color, age, shape, and so forth. Examples of properties are has-color, has-depth, has-interior-angles, etc[7]. The potential candidates for "properties/attributes" in the ontology are initially called "proto-characteristics." Proto-characteristic identification usually occurs concurrently with proto-kind identification because kinds are usually individuated on the basis of (some of) the properties/attributes that they exhibit. The listing of characteristics is a relatively straightforward task because characteristics are readily observable and are often measurable. The proto-characteristics are tagged with supporting source material (for traceability) and catalogued in the Proto-Characteristic Pool (Figure 3-11). The distinction between attributes/properties and kinds is not always clear, however, and guidelines to differentiate these concepts are given in Subsection 2.2.3. For illustration, a proto-kind **project task** can have five characteristics: 1) duration, 2)

[7]The difference between properties and attributes is described in Subsection 2.2.3.

earliest starting time, 3) latest starting time, 4) earliest finishing time, and 5) latest finish time. These attributes are catalogued in a Proto-Characteristic Pool [8], as shown in Figure 3-11.

Proto-Characteristic Pool

Project: Ontology of Project Planning — **Analyst :** P. Benjamin

Proto-characteristic	Proto-characteristic Name	Supported By	Supported By	Kinds Used In
PC #1	p-earliest starting time	SM #1	Hari	PK #8 PK #9
PC #2	p-latest starting time	SM #1	Hari	PK #8 PK #9
PC #3	p-earliest finish time	SM #2	Hari	PK #8 PK #9
PC #4	p-latest finish time	SM #2	Hari	PK #8 PK #9

Figure 3-11. Proto-Characteristic Pool

The following fields are used to describe the Proto-Characteristic Pool:

- **Characteristic #** The unique identifying number assigned to each characteristic is documented in this field. The characteristic number is important because it can be used as a means of traceability to the characteristic in the future.
- **Characteristic Name** The name of the characteristic is recorded in this field.
- **Supported By** The names of the team members responsible for the identification of the proto-characteristic are recorded in this field.

[8]When a proto-characteristic matures into an characteristic during ontology development process, the characteristic is recorded in a Characteristic Pool. The design of a Characteristic Pool is the same as that of a Proto-Characteristic Pool, as shown in Figure 4-9.

- **Kinds Used In** This field consists of the list of all kinds that possess this characteristic.

3.4.4 The Role of IDEF5 Schematics in Ontology Visualization

The IDEF5 languages are used during the invocation of the IDEF5 procedure to assist with the development and visualization of the ontology. The purpose of IDEF5 schematics is to serve as an aid for the construction of ontologies; they are not the primary representational medium for storing the ontologies (refer Subsection 4.1.2 for more details). The elements of the Schematic Language[9] that are used to illustrate the IDEF5 procedure in this section are as follows: 1) A kind is represented by a circle containing a label, 2) A first-order relation is represented either by an arrow with a label or by a rectangle with rounded corners containing a label, and 3) A second-order relation is represented by an arrow with its arrowhead at its back end. The use of Classification Schematics, Relation Schematics, and Composition Schematics are briefly illustrated in this section.

3.4.5 Using Classification Schematics for Ontology Development

The following example demonstrates how a Classification Schematic feature helps visualize the **subkind-of** relation between different kinds in a domain. A source statement "plans may be broadly classified as project plans, process plans, and manpower plans," can be considered. Suppose that based on this source statement, team members individuated kinds as **plan**, **project plan**, **process plan**, and **manpower plan** during data analysis. A Classification Schematic is designed to explicitly display the **subkind-of** relation between various proto-kinds/kinds in a domain, as shown in Figure 3-12.

[9]The IDEF5 Schematic Language is described in detail in Subsection 4.1.

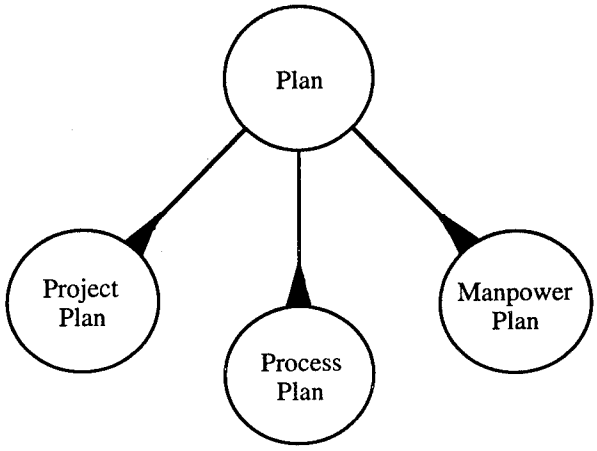


Figure 3-12. Classification Schematic

3.4.6 Kinds Versus Properties

A common problem in ontology development is distinguishing between kinds and properties. Properties, by definition, are characteristics that hold of kinds. An example of a property may be (an object) **being red**. However, in some circumstances, it may be useful to objectify (make objects of) the properties and treat them as kinds in their own right. To illustrate the nature of the problem, the tolerance of a mechanical part used to make an assembled product can be considered. For the manufacturing engineer performing the assembly, the tolerance is simply a property that must hold for all parts supplied to him. If the tolerance does not hold, it is rejected; otherwise, it is accepted. For the part designer, however, tolerances are of greater significance. Thus, the designer finds it useful to classify different kinds of tolerances (dimensional tolerance, positional tolerance, geometrical tolerance, etc.) and to study the characteristics of each kind of tolerance. The designer will, thus, find it useful to model tolerance as a kind rather than as a property. In general, the decisions between kinds and properties are a function of the granularity (detail) level of the ontology development effort. The level of granularity is significantly influenced by the purpose, viewpoint, and context of the project (see Subsection 3.1.2).

3.4.7 Coining Terms

Closely associated with the discovery of proto-kinds is the task of unambiguously identifying each of these proto-kinds; in practical terms,this means that each individuated candidate kind must be bestowed a name. Giving names in ontology development is not as trivial a task as it may first appear. Names are used as referential pointers to the real world individuals being described. They must, thus, connote a meaning that closely mirrors the individuals that are referenced. For many real world individuals (especially the phenomenological naive ones), widely accepted names

already exist and will be used as such. For example, domains of certain engineered products, such as automaking, have hardened and have fairly stable ontologies. (On the other hand, ontologies of new technological fields, such as "virtual reality," are in constant flux.) It may be necessary, at this stage, to invent new names for certain individuals. This is often true for new conceptual groupings of individuals, as illustrated in Figure 3-13. The strategy suggested in IDEF5 is to coin a term to describe the individual and record the fact that the individual's name was coined rather than discovered to avoid later confusion.

Conceptual objects are often discovered during interviews with domain experts. For example, during development of a fastener selection expert system for an automobile manufacturer, a domain expert had conceptualized a set of systems for fastener usage in various parts of a car body. These systems were not recorded in the literature, and the expert had never given them names. Rather, he carried the ideas with him internally and used them without names. The term **world class systems** was coined to describe these conceptual ideas (see Figure 3-13).

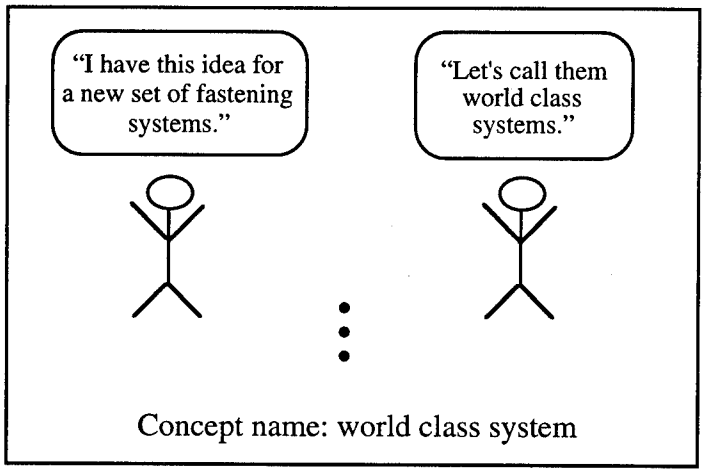


Figure 3-13. Coining Terms

The Proto-Kind Pool is provided in IDEF5 to record proto-kinds (see Figure 3-9). As seen in Figure 3-9, each proto-kind is tagged to the supporting source document for traceability purposes. As a further aid to identifying proto-kinds, the IDEF5 ontology library (see Appendix B) provides a catalog of generic kinds that commonly occur in engineering and business domains. These kinds may be used for effectively organizing the captured ontology knowledge about different kinds of objects and phenomena.

3.4.8 Other Guidelines

Other useful guidelines for developing proto-kinds are as follows.

- Identify and record special cases. Special cases are instances of objects that do not seem to fit the pattern of other instances of the object. For example, a particular fastener in a handbook may have a special heat resistant property not present in other fasteners. All special cases are catalogued separately in the source material catalog.

- Group objects together to form new kinds and categories, wherever appropriate. It may be helpful to abstract away from a group of objects and form new proto-kinds by extracting the properties from the object group. Such abstractions often serve to enhance the conceptual clarity of the ontology.

- Group the objects to isolate systems and subsystems. Systems are collections of objects that fulfill a common purpose. Systems may not have any distinguishing characteristics but often provide a useful framework for organizing knowledge about relations.

- Use the IDEF5 classification and other relation schematics to aid the conceptual analysis of kinds. The classification schematics and the composition schematics are particularly useful in the development of kinds.

Figure 3-14 summarizes some of the key activities associated with developing proto-kinds.

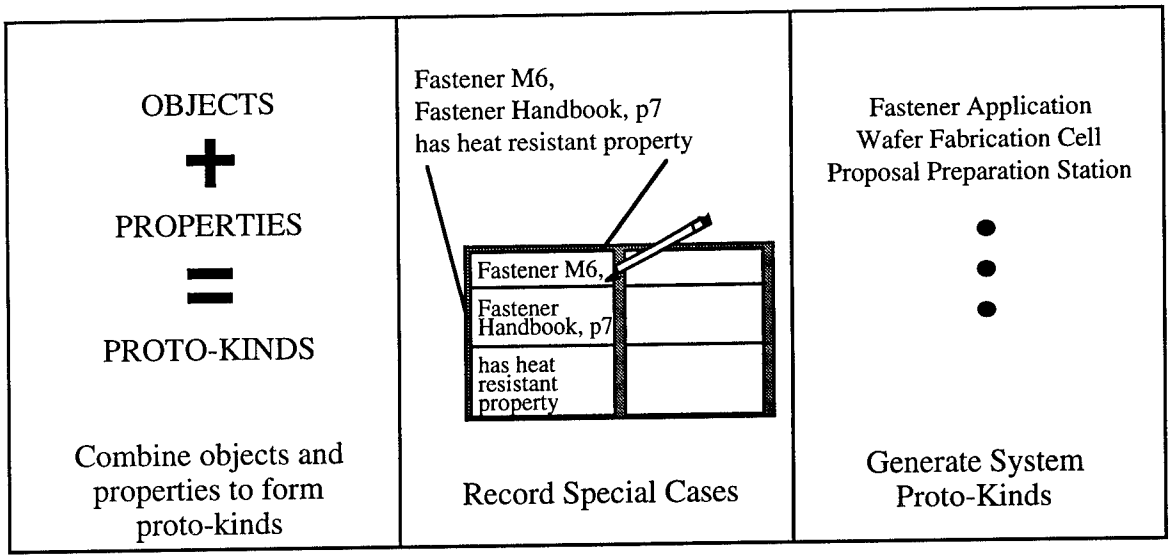


Figure 3-14. Developing Proto-Kinds

3.4.9 Develop Proto-Relations

The objective of this task is to identify and characterize the proto-relations between the proto-kinds. A proto-relation is the result of a preliminary attempt at individuating a relation, and it expresses associations between the proto-kinds. The identification and characterization of relations is often the most difficult part of knowledge acquisition. The identification of proto-relations refers to the activity of recognizing the existence of, or becoming attuned to, a particular proto-relation in the domain. Characterization follows identification and refers to the activity of identifying and specifying the properties of a proto-relation in a manner that will allow the relational knowledge to be used for making useful inferences at some time in the future. Thus, recognizing that a tool post is On-top-of the lathe bed is the act of discovering and asserting its existence and giving it a name. Characterizing it will involve making assertions such as **the On-top-of relation is transitive**.

Several mechanisms are provided in IDEF5 to guide the relation knowledge acquisition process. They are:

- The IDEF5 Statement Pool is the richest source of information for relation discovery and characterization. Source statements assert the existence of relations either implicitly or explicitly.

- The IDEF5 Relation Library provides a catalog of relations that commonly occur in business and engineering domains. These libraries can be reused and tailored to the requirements of particular ontology development efforts within a wide range of domains.

- The IDEF5 Relation Schematics (including Composition Schematics) facilitate the display of relations in a graphical form.

- The IDEF5 Elaboration Language, which provides a structured text format for capturing complex relation knowledge at any level of complexity, can express everything that can be recorded using the Schematic Language; it can also express knowledge that is beyond the scope of the Schematic Language. For example, the expression z = a + b + c + d, where z, a, b, c, and d are integers, can only be expressed in the IDEF5 Elaboration Language[10].

---

[10]The IDEF5 Elaboration Language is described in Section 4.2.

The development of proto-relations will involve the following activities.

- Record meaningful associations between proto-kinds. Such meaningful associations often indicate the existence of proto-relations. A useful source for extracting such associations is the Source Statement Pool (Subsection 3.2.3.3). A Proto-Association Chart is a two-dimensional matrix with relevant proto-kinds listed on both axes. An X is marked in cells where the existence of a possible proto-relation is indicated, as shown in Figure 3-15.

- Categorize the proto-relations as being system-accidental or system-essential and recall that system-essential relations have to necessarily hold, given the existence of instances of the participating kinds.

- Identify the properties of the proto-relation. The IDEF5 relation library and the IDEF5 elaboration language are used to facilitate this process.

- Examine the nature of the participating proto-kinds. Relations often have restrictions on the types[11] of arguments. For example, the assertion **A Reports-to B** implies that both A and B refer to instances of people, or to instances of organizational roles, or to a combination of an organization role instance and a person instance. Thus, knowledge of the participating proto-kinds will focus attention on a more restricted set of possible relations that may exist between instances of these proto-kinds.

[11]The term "type" is used here in a general sense. See Subsection 2.1.2 for a discussion of kinds and types.

Kinds / Kinds	K1	K2	K3	K4	• • •	Kn
K1			X			
K2						
K3	X			X		
K4			X			
• •						
Kn						

Note: Proto-kinds are denoted K1, K2, ... , Kn.

Figure 3-15. Structure of a Proto-Association Chart

The proto-relations that are identified are recorded in the Proto-Relation Pool[12], as shown in Figure 3-16.

Proto-Relation Pool				
Project: Ontology of Project Planning			**Analyst:** P. Benjamin	
Proto-Relation #	**Proto-Relation Name**	**Supported By**	**Participating Kinds**	**Comments**
PR #1	p- Determines	SS #2	PK #7, PK #9	
PR #2	p- Specifies	SS #2	PK #7, PK #8	

Figure 3-16. Proto-Relation Pool

- **Proto-Relation #** The unique identification number assigned to the proto-relation is recorded in this field. This field is important to enhance traceability of the proto-relation that is individuated by a member of the ontology development team.

[12]When a proto-relation matures into a relation during ontology development process, the relation is recorded in a Relation Pool. The design of a relation pool is same as that of Proto-Relation Pool, as shown in Figure 3-17.

- **Proto-Relation Name** The name of the proto-relation is documented in this field.
- **Supported By** This field involves source data items (terms, statements) that support the individuation process of the relation during the ontology development process.
- **Participating Kinds** The set of all kinds between which this relation holds is documented in this field, which is important because it helps in characterizing the relation itself.
- **Comments** Any additional descriptive remarks about the proto-relation important for future reference should be recorded in this field.

Each proto-relation has a Proto-Relation Specification Form (Figure 3-17), in which a brief description of the proto-relation, examples of use, and other relevant comments can be documented.

Proto-Relation Specification Form	
Project: Ontology of Project Planning	**Analyst:** P. Benjamin
Proto-Relation #: PR #1	
Proto-Relation Name: Determines	
Description: To fix conclusively or authoritatively.	
Examples of Use: A project planner determines all the precedence constraints that must be maintained in the system.	
Comments: Individuated by B. Caraway	

Figure 3-17. Proto-Relation Specification Form

3.4.10 Role of Relation Schematics in Ontology Development

The IDEF5 Relation Schematics allows ontology developers to visualize and understand relations between relevant proto-kinds or kinds in a domain. Consider the source statement: **a manpower planner develops a manpower plan in which resources are allocated to perform the**

required activities. Based on this source statement, suppose that the ontology development team individuated the following kinds: **manpower planner**, **manpower plan**, **resource**, and **activity** . The relations between these kinds are visualized using the relation schematic shown in Figures 3-18 or, alternatively, Figure 3-19.

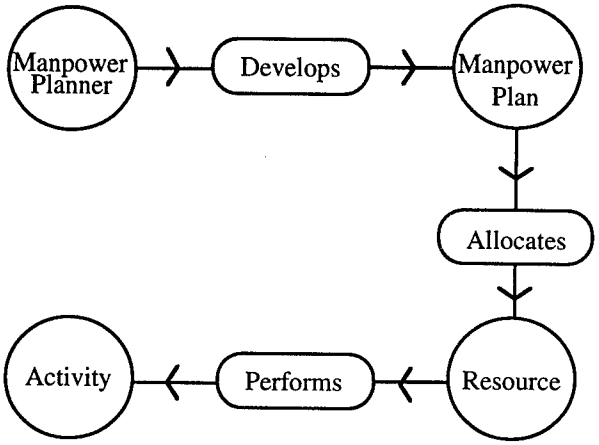


Figure 3-18. First-order Schematic

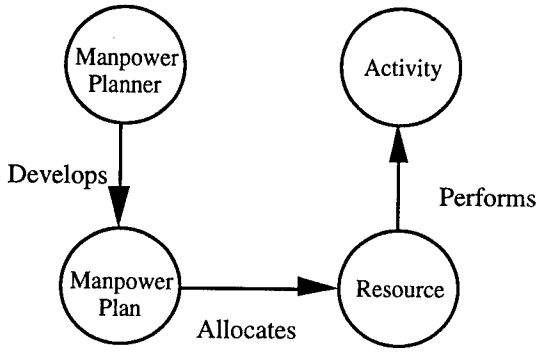


Figure 3-19. Alternative Syntax for the Schematic in Figure 3-18

3.4.11 Role of Composition Schematics in Ontology Development

The IDEF5 Composition Schematics allows ontology team member(s) to visualize **part-of** relations between relevant proto-kinds or kinds in a domain. Consider the following source statements: **A project plan is comprised of project goals, an activity list, and the manpower allocation plan** and **The activity list includes a major activity list and a minor activity list**. Based on these source statements, team members individuate the following kinds: **project plan**, **project goal**, **activity list**, **manpower allocation plan**, **major activity list**, and **minor activity list**. Figure 3-20 illustrates the use of a composition schematic to display **part-of** relations in this (project planning) domain.

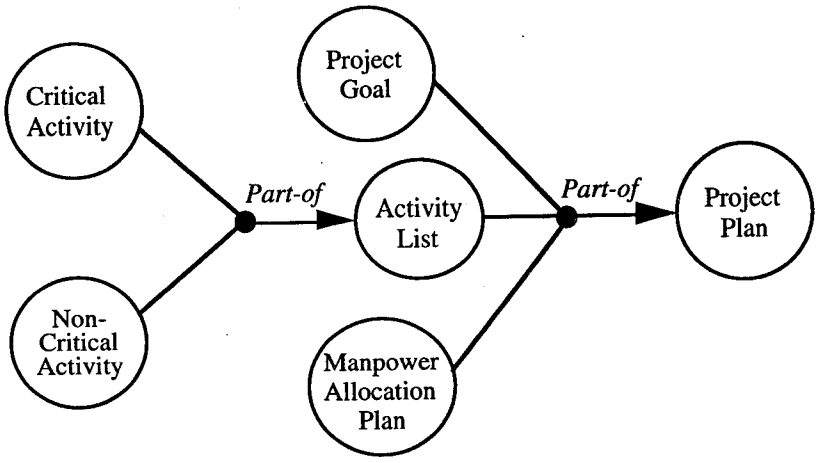


Figure 3-20. Composition Schematic

3.5 Refine and Validate Ontology

The objective of this phase of ontology development is to refine the proto-characteristics, kinds, and relations, and to affirm their authenticity by converting them to properties/attributes, kinds, and relations, respectively. The refinement process is essentially a deductive validation procedure. The ontology structures are "instantiated" (tested) with actual data, and the result of the instantiation is compared with the ontology structure. If the comparison produces any mismatch, every such mismatch must be adequately resolved. Refinements (if any) to the initial ontology are incorporated to obtain a validated ontology.

3.5.1 Kind Refinement Procedure

The kind refinement procedure is summarized in the following (roughly, but not necessarily, sequential) steps:

- Make instances of the proto-kinds. The examples may be constructed from the available source data (source data log); otherwise, new data must be gathered for the purpose of constructing these examples. The examples must be reasonably representative, with at least one exception case included, if possible. Each of the proto-kind instances created is populated with properties and/or attributes (this may involve converting proto-characteristics to properties/attributes). Classification schematics and kind characterization forms are used to support the kind instantiation process.

- Record information that cannot be recorded in the kind instances, determining whether this additional information is really necessary, and, if so, refining the structure of the kind to include the information.

- Check whether two instances of the same kind have different defining properties, and in such cases, check whether the viewpoints are different. If not, the inconsistencies will have to be resolved by refining the ontology.

- Convert the proto-kinds (along with their proto-properties) to kinds after all the kind instances have been validated using Steps 1 through 3. The validated kinds are listed in the Kind Pool.

A Kind Specification Form (Figure 3-21) is designed to document all relevant features of a kind after it is promoted from a proto-kind.

Kind Specification Form	
Project: Ontology of Project Planning	**Analyst:** P. Benjamin
Kind #: K #1	
Kind Name: Resource	
Description:	
Attributes: 1. Has status : active/idle (Defining) 2. Has a unique identification number (Non-defining) 3. Has a unique label (Non-defining)	
Defining Properties: 1. Is necessary for performing task/activity efficiently	
Properties: 1. Identification number is R #857 (Accidental) 2. Label is "Capstan Lathe #4" (Accidental)	
Relation the Kind Participates in:	
Comments:	
Elaboration Language Specification:	

Figure 3-21. Kind Specification Form

The various fields in a Kind Specification Form are briefly described as follows.

- **Kind #** The unique identifier of a kind is recorded in this field. The kind number is important because it provides traceability to the kind.

- **Kind Name** The name of the kind is recorded in this field. This kind may have been promoted from a proto-kind status by team members during the ontology development process.

- **Description** A brief description of the kind is documented in this field.
- **Attributes** All the attributes of the kind are documented in this field.
- **Defining Properties** All the properties of the kind are recorded in this field.
- **Properties** All non-defining properties are recorded in this field. Each non-defining property may be categorized as an essential or accidental property and tagged to the stated property.
- **Relations the Kind Participates In** All the relations in which the kind participates are documented in this field.
- **Comments** Any additional descriptive remarks about the kind important for future reference should be recorded in this field.
- **Elaboration Language Specification** The IDEF5 Elaboration Language specification of the kind should be documented in this field. (The IDEF5 Elaboration Language is described in Subsection 4.2.)

3.5.2 Relation Refinement Procedure

The relation refinement procedure is summarized in the following (roughly, but not necessarily, sequential) steps:

- Make instances of the proto-relations. The examples may be constructed from the available source data (source data catalog); otherwise, new data must be gathered for this purpose. The IDEF5 Relation Schematics and the IDEF5 Relation Characterization forms are used to aid the instantiation and validation procedure.
- Compare the properties of each of the relation instances with the properties identified in the IDEF5 description, thus resolving any mismatches. Moreover, missing relation properties should be checked for and added, if needed.
- Sample instances of selected system essential relations and examine whether two or more instances of such relations are incompatible. For example, one system-essential relation may say that a fastener must have a sealant, and another may say that it cannot have a sealant. Such inconsistencies may be either due to hidden viewpoint differences not recorded in the ontology or to differing viewpoints. Incompatibilities that occur because of differing viewpoints may be resolved by splitting the focus

relation into different relations, one for each viewpoint. Otherwise, a consensus must be reached to resolve the incompatibility through discussions with the domain expert.

- Detect new relations discovered by example that were not captured in the ontology and add such relations to the ontology.

- Make instances of *inference sequences* using the relation properties, if appropriate. For example, if it is recorded that relation **R** is transitive, instances **x**, **y**, and **z** of kinds **A**, **B**, and **C** respectively such that **xRy** and **yRz** are both true must be found. The transitivity of **R** must be validated by checking whether **xRz** is true.

- Convert the proto-relations to relations, after all the relation instances have been validated using Steps 1 through 5, and record the validated relations in a Relation Pool.

The Relation Specification Form (Figure 3-22) is designed to facilitate the characterization of relations. Notice that the Elaboration Language statements are also recorded on this form.

Relation Specification Form
Relation #: R #1
Relation Name: Determines
Description: To fix conclusively or authoritatively.
Arguments: Project Planner, precedence constraints
Examples of Use: A project planner determines all the precedence constraints that must be maintained in the system.
Comments: This relation is individuated by B. Caraway.
Elaboration Language Specification:

Figure 3-22. Relation Specification Form

- **Relation #** The unique identification number assigned to the relation is recorded in this field. This field is important to enhance traceability of the relation that is individuated by a member of the ontology development team.

- **Relation Name** The name of the relation is documented in this field.

- **Description** This field records a brief explanation of the relation.

- **Arguments** The set of all kinds between which this relation holds is documented in this field, which is important because it helps in characterizing the relation itself.

- **Examples of Use** A set of sentences, that provides a basic idea of how the relation holds between kinds, can be recorded in this field.

- **Comments** Any additional descriptive remarks about the relation important for future reference should be recorded in this field.

- **Elaboration Language Specification** The IDEF5 Elaboration Language specification of the relation should be documented in this field. (The IDEF5 Elaboration Language is described in Subsection 4.2.)

4 The IDEF5 Ontology Languages

A domain ontology is a detailed characterization of "what there is" in a given application domain. Such characterizations must, of course, be given in some language; hence, languages play an important role in the ontology capture process. More specifically, languages for ontology capture are important for two reasons.

- They provide a medium for capturing and storing knowledge.
- They provide a format for displaying the acquired knowledge.

Representational structures that are rich in expressive power are important for ontology because previously acquired knowledge is often used to guide the process of acquiring additional knowledge.

Because it cannot generally be determined a priori what sorts of representational structures will be needed to capture the ontology of a given domain, languages for ontology need to be expressively very rich. However, if the method of ontology capture is to be usable, its representational structures must be intelligible to ontology developers. The ease of use of a language is determined by its "look and feel" and by how well it supports the cognitive activities of the ontology development process. The ontology languages must have a synergistic relationship with the ontology development procedure (Section 3). That is, the languages must support the use of the procedure and the procedure must support the use of the languages.

The purpose of this section is to describe the ontology capture languages. IDEF5 has two languages.

- **The IDEF5 Schematic Language** This language is the graphical component of the IDEF5 languages. It provides visual assistance in the ontology capture process and facilitates communication.[13]

13The IDEF5 Schematic Language is supplemented by two important forms: 1) the Kind Specification Form and 2) the Relation Specification Form. These forms allow the ontology developer to record the ontology in a textual form. It permits ontology characterization at a greater level of detail than what is possible in a graphical form. The IDEF5 forms are also designed to facilitate the recording of IDEF5 Elaboration Language statements. The IDEF5 forms are described in Subection 3.1.4. The IDEF5 Elaboration Language is presented in Section 4.2.

- **The IDEF5 Elaboration Language** This language is a structured textual language and has the full power of first-order logic.

The two IDEF5 languages complement and supplement each other. The Schematic Language is somewhat restricted in expressive power. However, the graphical structures of this language make it intuitive and easy to use. The IDEF5 Elaboration Language, on the other hand, is an expressively rich textual language. In particular, it can express everything that can be expressed in classical first-order logic. However, the structured text syntax requires a higher level of proficiency for its effective use.

4.1 The IDEF5 Schematic Language

Essentially, an ontology identifies and organizes the relevant kinds and individuals, their properties, and the network of relations between them within a specific application domain. The IDEF5 Schematic Language provides a variety of graphical constructs to assist in the construction of ontologies. In the following subsections, a summary of the IDEF5 Schematic Language lexicon is first presented. The use of these representational "building blocks" to develop diagrams, or *schematics*, is then described with the help of illustrative examples.

4.1.1 The Schematic Language Lexicon

The central primitive symbols of the IDEF5 Schematic Language are shown in Figure 4-1.

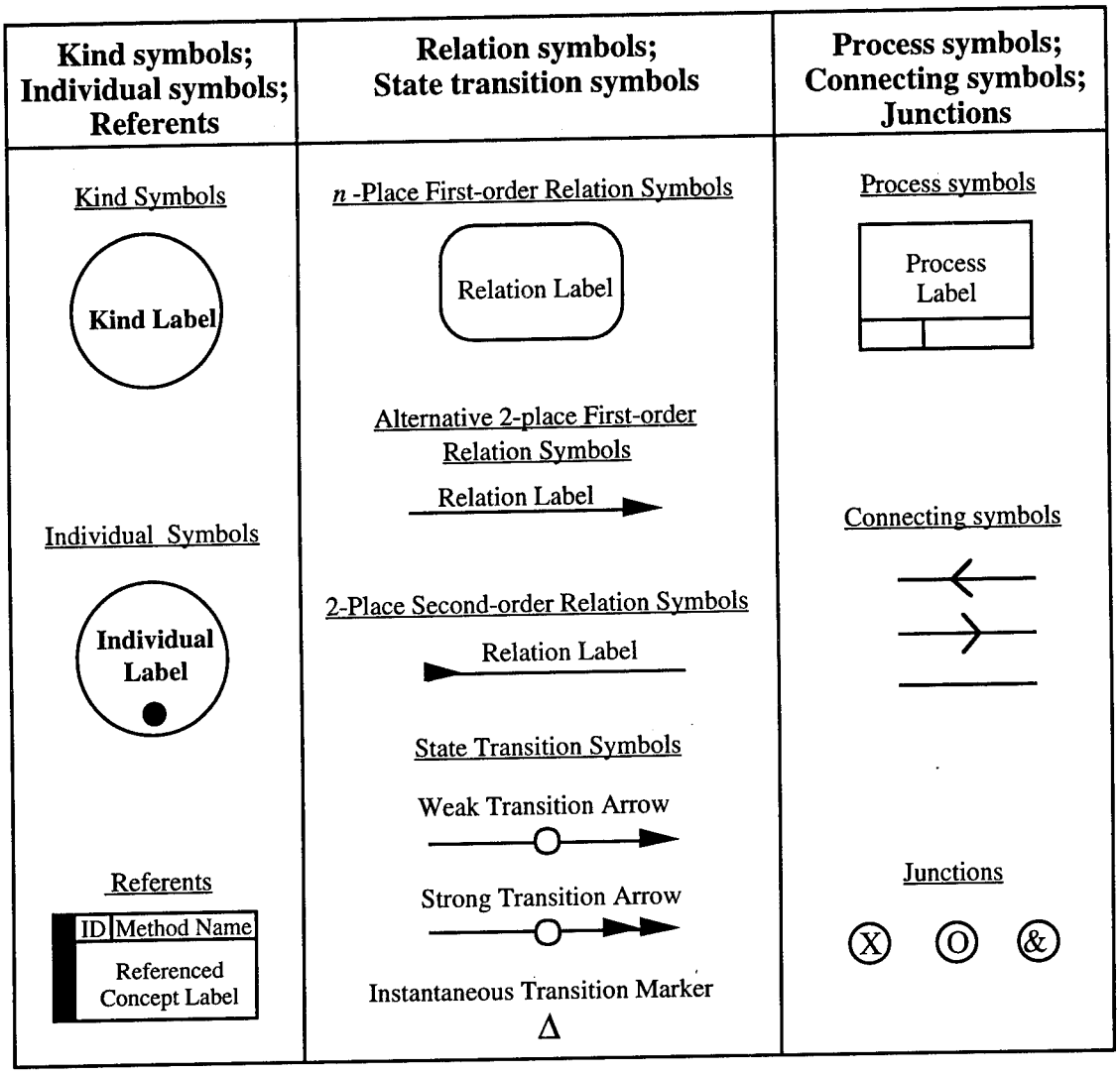


Figure 4-1. Basic IDEF5 Schematic Language Symbols

The *basic lexicon* of the IDEF5 Schematic Language (see Figure 4-1) consists of the following:

- **Kind Symbols** A kind is represented by a circle containing a label. The label in a kind symbol should be either identical with the name of the kind given in the associated specification form for the kind (Subsection 4.1.4), or in shorthand form of the name.

- **Individual Symbols** Labeled circles that also include a small, filled-circle (as shown in Figure 4-1) represent specific individuals identified in an ontology. The label should be unique within the ontology.

- **Referents** An IDEF5 Referent is an artifact used to make reference to a concept in any other IDEF method. Referent rectangles have the following information items.

  1) **Referenced Concept Label** This is the label of the concept (within an IDEF model) that is being referenced. For example, an activity kind may reference an activity label in an IDEFØ model.

  2) **ID** This is the identification label for the Referent.

  3) **Method Name** This is the name of the IDEF method that is being referenced.

- **Relation Symbols** Rectangles with rounded corners signify first-order n-place relations (i.e., relations that hold between first-order individuals). In the case of 2-place relations, a labeled arrow can be used instead of a rectangle. Arrows with their arrowheads at the back represent second-order relations (i.e., relations that hold, not between individuals, but between kinds or between individuals and kinds). There are no symbols in the IDEF5 Schematic Language for *n*-place higher-order relations, as experience has shown their presence in ontologies to be rare. If necessary, however, such relations can always be added to an ontology via the IDEF5 Elaboration Language (see Subsection 4.2). Because of the importance of the first-order **part-of** relation in ontology, the distinguished label part-of is included in the IDEF5 Schematic Language.

  Every relation symbol includes a label connoting the relation represented. As with kind symbols, the label should be either identical with the name of the relation given in the associated specification form for the relation or in shorthand form of the name. Like **part-of**, the distinguished labels instance-of and subkind-of are provided for the second-order relations **instance-of** and **subkind-of**.

- **State Transition Symbols** There are two types of state transition links provided by IDEF5 schematic languages: 1) an arrow with an open circle at the center of the arrow that represents weak transition and 2) a double headed arrow with an open circle at the center of the arrow that represents strong transition.

  An instantaneous transition marker "Δ" is provided by the IDEF5 Schematic Language to represent the state transition that occurs over a period smaller than the smallest time unit recognized in the context being modeled.

- **Process Symbols** Rectangles with square corners and a line near the bottom indicate process kinds (i.e., general, repeatable states of affairs, as discussed in Section 2). Their syntax and informal semantics are provided in Subsection 4.1.5.

- **Connecting Symbols** Connecting symbols are a type of arrow used to connect kinds to first-order relations. Their syntax is discussed in Subsection 4.1.5.

- **Junctions** Junctions are simply symbols that represent boolean operators. Their syntax and semantics are discussed in Subsection 4.1.5.

4.1.2 IDEF5 Schematics and their Interpretation

In this subsection, the various diagram types, or *schematics*, that can be constructed in the IDEF5 Schematic Language are presented. The purpose of these schematics, like any representations, is to carry information. Thus, semantic rules must be provided for interpreting every possible schematic. Following the usual approach, these rules will be provided by providing rules for interpreting the most basic constructs of the language, which can then be applied recursively to more complex constructs. However, the character of the semantics for the Schematic Language differs from the character of the semantics for other graphical languages. More specifically, each basic schematic is provided only with a *default* semantics that can be overridden in the Elaboration Language (see Subsection 4.2). The reason for this is that the chief purpose of the Schematic Language is to serve as an *aid* for the construction of ontologies; they are not the primary representational medium for storing them. That task falls to the Elaboration Language. The Schematic Language particularly useful for the construction of *first-cut* ontologies in which the central concern is to record in a rough way the basic kinds of things that exist in a domain, their characteristic properties, and the salient relations that can be obtained between objects of those kinds and between the kinds themselves. Consequently, the basic constructs of the Schematic Language are designed specifically to capture this type of information. However, the default semantics may not accurately capture the desired information. In such a case, a user can explicitly override the default semantics in the Elaboration Language.

However, the precise character of the properties of, and relations between, objects of various kinds is often quite different from one case to another: Does the relation hold between every pair of instances? Some instances? Must it hold between two instances if it holds between them at all? Therefore, to assign a semantics that enforces just one of these semantic possibilities with respect to a given construct rules out other legitimate possibilities and, hence, limits the flexibility and usefulness of the language in the construction of an ontology, because any of the possibilities might be present in the domain under consideration. The idea behind the default semantics, for

each construct, is to express what experience has shown to be the most useful meaning as a default but also to allow revision or further specification of this meaning in the Elaboration Language, should the need arise.

This issue raises an obvious question: Why not simply endow the Schematic Language with the power to express these subtle semantic distinctions directly? Why not simply adopt a graphical representation language like Semantic Nets (SNs — in its more sophisticated guises) or Conceptual Graphs (CGs) that have the full expressive power of a first-order language?[14] Why not adopt a graphical language that is intended to be complete in itself, without need of any further supplementation by a nongraphical language, instead of one that falls far short of the full expressive power of a first-order language?[15] The central answer is again rooted in the fact that the Schematic Language is designed only for the construction of first-cut ontologies and the easy browsing and rapid addition of new information to existing ontologies. It was designed to streamline the process of ontology construction and evolution, not to be the central representational medium for ontology. Among the central considerations behind its design, then, was to give domain experts largely unfamiliar with full first-order languages and the subtleties of modality and quantification an intuitive interface for entering basic ontology information. For this purpose, a graphical language is ideal. However, beyond relatively simple pieces of information about kinds and their characteristic properties and relations, graphical languages have no advantage over standard linear languages in either learnability or usability; indeed, in the eyes of many experts, graphical representations of more complex information are substantially more cumbersome than their standard linear counterparts. There is, thus, no reason not to carry out refinements of an ontology in the Elaboration Language directly. That is at any rate much more efficient with respect to the development of an automated ontology support tool: a graphical language is not in itself computer processable. Hence, if the information expressed in such a language is to be manipulated, queried, updated, and so forth, it needs to be compiled into a more standard computer processable form. The Elaboration Language, however, is processable as it stands, and hence,

---

[14]That is, loosely, for any given SN or CG language L, there is a mapping that takes any SN or CG A of L into a set of sentences in a first-order language with the same meaning. That is, A and the set will be true of the same models and vice versa (for finite sets of sentences). For SNs, see S. Shapiro, "The SNePS semantic network processing system," in N. Findler, *Associative Networks: Representation and Use of Knowledge by Computers*, New York, Academic Press, 1979. For CGs, see J. Sowa, *Conceptual Graphs: Information Processing in Mind and Machine*, Menlo Park, CA, Addison-Wesley Publishing Co., 1984.

[15]More exactly, with regard to its default semantics, it is, very roughly, equivalent to a first-order language that permits only strings of existential quantifiers.

once again, there is no advantage in using an expressively equivalent graphical language for ontology refinement.s

4.1.2.1 Basic First-Order Schematics

This subsection presents the syntax and default semantics for the basic first-order schematics of the language. The simplest of these are schematics involving 2-place, first-order relations.

4.1.2.1.1 2-Place Relation Schematics

IDEF5 schematics are constructed by putting together the basic IDEF5 graphical symbols in different ways. The *basic* IDEF5 schematics are the smallest diagrams that can be so constructed. The syntax of the basic schematics is quite simple: they are obtained by connecting circles with relation and proto-relation symbols.

The most common construct of this sort involves connecting two kind symbols with a first-order relation symbol, as in Figure 4-2.

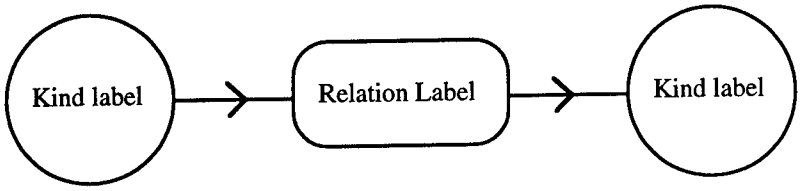


Figure 4-2. General Form of a Basic First-Order Schematic

Such diagrams need a default semantics (i.e., an accepted meaning that can be assumed in the absence of any further clarification in the Elaboration Language). For this purpose, we considered the concrete example in Figure 4-3.

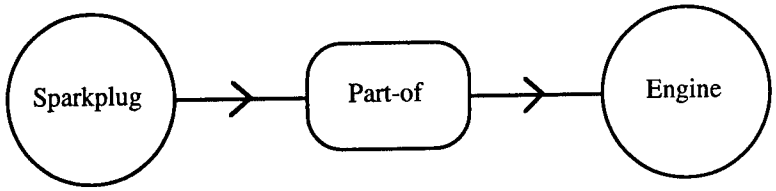


Figure 4-3. Example of a Basic First-Order Schematic

What, exactly, should the default meaning of this construct be? It is very important to note that it does not mean that the kind **sparkplug** is a part of the kind **engine**, for **part-of** in IDEF5 is a first-order relation that holds between first-order physical objects, and, in particular not between abstract, higher-order objects like kinds. Thus, Figure 4-3 must be about *instances* of the kinds in

question–the individuals that exemplify those kinds. Given that there are a variety of possible meanings that might be assigned to Figure 4-3, the following specifications are made.

(1) Every sparkplug (in the domain in question) is a part of every engine (in the domain in question).

Clearly that is not even physically possible, in general: sparkplug can only be a part of one engine. A weaker reading is the following.

(2) Every sparkplug is a part of some engine.

This is a much more plausible reading, but as a default, it is still too strong. There may be loose sparkplugs in inventory or on the shop floor.

(3) Every engine has some sparkplugs among its parts.

This reading takes account of the possibility of loose sparkplugs, but now the problem is that there may well be engines lying about with the plugs removed, so this, too, is too strong.

(4) Some sparkplugs are parts of some engines.

This reading now accounts for both counterexamples, but again, as a default, it still might say too much. That is, Figure 4-3 might only be recording the way the domain in question *ordinarily* is, but not at all any way that it is at all times. For instance, the domain in question might be an ontology for an automobile factory that has temporarily shut down its engine fabrication division and is making only car bodies and, hence, which, for the time being, contains no engines at all. To accommodate such cases, a reading weaker than (4) is needed for the default semantics of Figure 4-3; specifically, the default semantics for schematics like the one in Figure 4-3 will be taken to indicate only what is possible, or *permissible*, in a certain domain vis-a-vis sparkplugs and engines relative to the **part-of** relation. Thus, more exactly, it will be taken as a default to mean only that, in the current ontology.

(5) Sparkplugs *can be* parts of engines.

That is, it is possible, or permissible, that a sparkplug (i.e., an instance of the kind **sparkplug**) be a part of an engine (i.e., an instance of the kind **engine**).[16] Therefore, the default semantics of

---

[16]Figure 4-3 is equivalent to the elaboration language statement (possibly (exists (?x ?y) (and (Sparkplug ?x) (Engine ?y) (Part-of ?x ?y)))).

schematics like Figure 4-3 should be viewed as analogous to type declarations in a program that gives the permissible arguments to certain functions without declaring precisely how those functions are instanced in a run of the program. In the same way, Figure 4-3 is essentially specifying permissible arguments for the **part-of** relation.

Note that the sense of possibility in question here is stronger than mere logical possibility. In the purely abstract sense, it is *logically* possible for almost anything to bear a given relation to anything else. It is logically possible, for example, that a sparkplug be part of an oddly designed computer. The sense of possibility in question here, though, is intended to reflect the actual nature of a given domain; it is intended to capture the way things can be in the domain, *other things being equal*, not how things could be, under any imagined circumstances. Indeed, generally speaking, in the evolution of an ontology, basic assertions will more generally reflect actual observations in the domain (i.e., generalizations of what has actually been observed in the domain: Specific sparkplugs have been observed to be parts of specific engines. These observations are then generalized in the form of Figure 4-3 and, for the reasons noted, given the semantics of (5). It should again be noted, though, that this is only a default semantics; stronger meanings for such diagrams (3), for instance can be imposed via the Elaboration Language.

As an alternative syntax for 2-place first-order schematics, it is permissible (and often preferable) to replace the two connecting symbols and the relation symbol with a single arrow labeled by the same relation label, as illustrated in Figure 4-4.

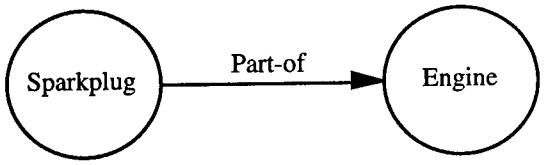


Figure 4-4. Example Illustrating Alternative Syntax for Basic First-Order Schematics

The direction of the arrow, as with the direction indicated by connecting symbols, corresponds to the natural English reading of sentences involving the kind labels and the relation label: **Sparkplug** can be **Part-of** an **Engine**.

4.1.2.1.2 Existential Schematics

The semantics of basic schematics like Figure 4-3 suggests a natural understanding of a kind symbol standing alone as in Figure 4-5:

Figure 4-5. An Existential Schematic

Specifically, this should be understood simply as the assertion that **Sparkplug** is a kind that can be instantiated in the domain, or more colloquially, that sparkplugs are among the things that one can expect to find in the domain. Again, this is a rather weak reading; it does not imply that there are, in fact, any sparkplugs in the domain, only that there *could* be, in the sense that the domain in question is appropriate for such things.[17] Stronger readings can, once again, be enforced in the Elaboration Language.

Free standing (first-order) relation symbols are also allowed, as in Figure 4-6.

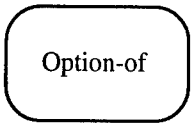


Figure 4-6. An Existential Schematic for a Relation

Analogous to Figure 4-6, Figure 4-5 asserts that the **Option-of** relation is one that can hold in the domain in question, or again, more colloquially, that one thing's being an option of another is something one can expect to find in the domain.

Because they assert the possible existence of objects of a certain kind in a domain, free-standing kind symbols like Figure 4-5 and its ilk will be known as *existential* schematics. Such schematics are useful insofar as they enable one to record that certain kinds have been observed in a given domain without requiring any further information about the relations such objects stand in with other objects.

---

[17]Thus, the sense of possibility in question, the sense of "could be" here, is somewhat stronger than mere logical possibility. It is *logically* possible that virtually *any* kind be instantiated in *any* domain. However, in most cases this will require extraordinary or unusual means outside the nature and function of the agents and mechanisms in the domain. For example, it is, of course, logically possible for an employee to bring a baseball into a semiconductor fabrication domain, thus instantiating the kind **baseball**. But it would, presumably, not be part of that agent's role in the domain to do so. Hence, in the sense in question, the kind **baseball** is not a kind that could be instantiated in the domain.

4.1.2.1.3 *n*-Place First-Order Schematics

The semantics for first-order schematics involving 2-place (first-order) relation symbols generalizes to schematics involving *n*-place relation symbols. So, for example, Figure 4-7 indicates only that an instance of the **conveys-to** relation can involve a **conveyer**, a **car body**, and a **paint primer vat**.

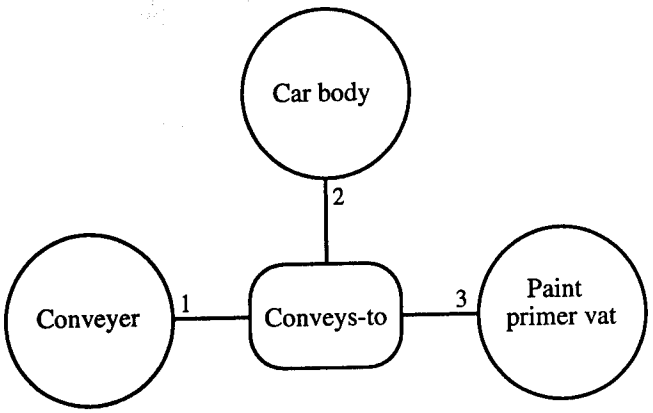


Figure 4-7. Example of a Basic 3-Place First-Order Schematic

The numbers (optionally) attached to the spokes here generalize the arrows on connecting symbols in the 2-place case. Specifically, they indicate that **conveyer**, **car body**, and **paint primer vat** are to be associated with the first, second, and third argument places of the **conveys-to** relation, as they occur in the natural English reading of the label: **conveyer** conveys a **car body** to a **paint primer vat**.

As in the 2-place case, the relation symbol can be omitted and labeled links can simply be used, as in Figure 4-8.

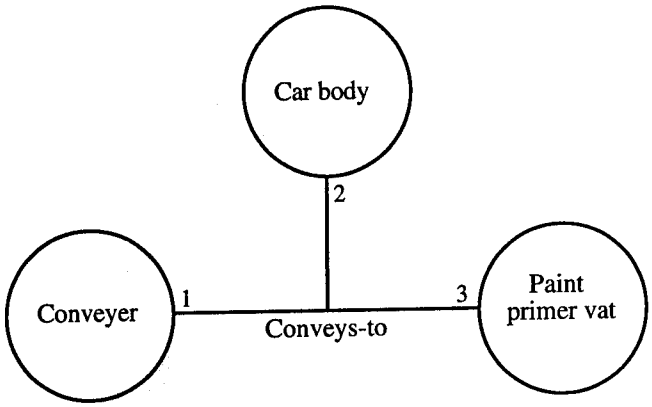


Figure 4-8. Alternative Syntax for Figure 4-7

In this document, this notation will generally be preferred.

Though they are somewhat uncommon, relations of arity four and greater can be expressed in a similar fashion. To illustrate, the general form of 4- and 5-place relation schematics are shown in Figure 4-9 (though the placement of the relation label and the numbering of the kind symbols can vary); if desired, of course, a rectangular relation symbol containing the relation label can alternatively be placed near the middle of the diagram.

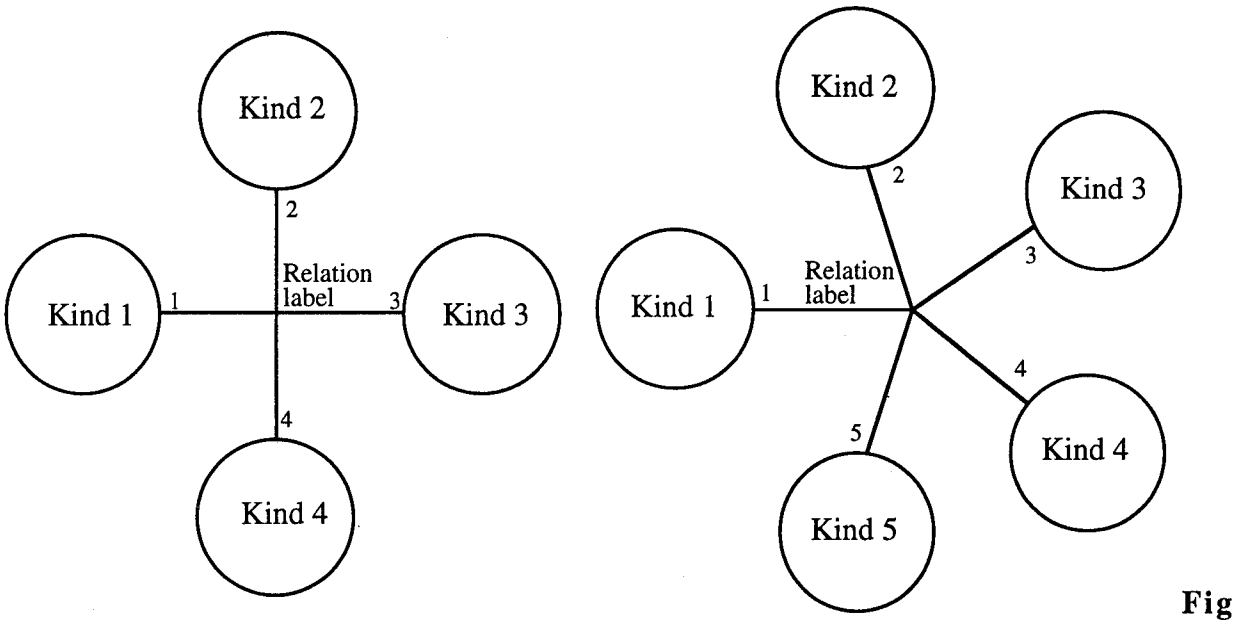


Figure 4-9. General Form of 4- and 5-Place First-Order Schematics

4.1.2.1.4 Using Individual Symbols

The use of individual symbols eliminates some of the indefiniteness of the schematics in Figure 4-8. For instance, the situation depicted by Figure 4-8 permits multiple paint primer vats. However, it might be desirable in some situations to focus on, say, one particular vat, and hence to represent it explicitly by an individual symbol as in Figure 4-10.

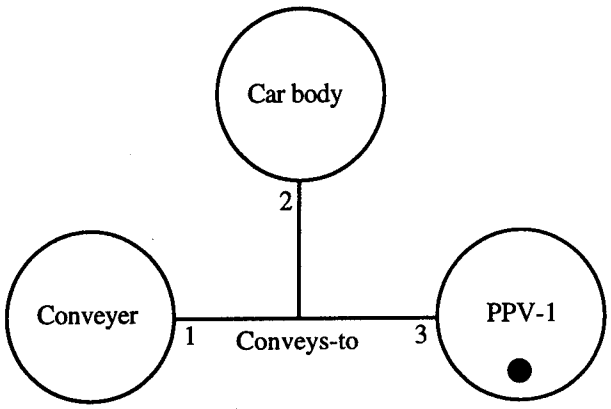


Figure 4-10. Example Illustrating the Use of an Individual Symbol

This schematic now expresses that a conveyer can convey a car body to the particular primer vat PPV-1, indicated by the individual symbol, a more definite proposition than the one expressed in Figure 4-8.

Indefiniteness is eliminated completely if only individual symbols are used. Thus, the schematic in Figure 4-11 is taken to express that the particular car body **CB-J27-S121** is (as opposed to only *can be*) at some time conveyed by conveyer **Conv-2** to the paint primer vat **PPV-1**.[18]

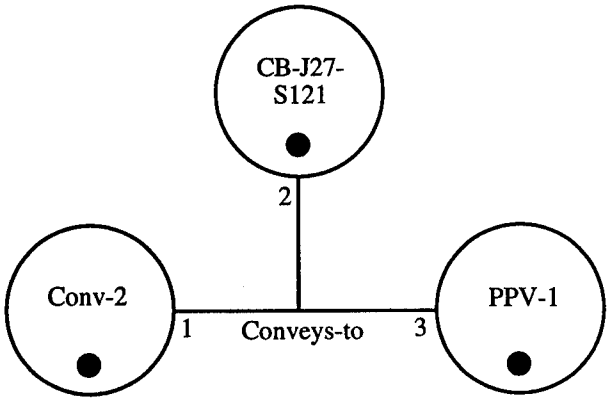


Figure 4-11. A Fully Particularized Example

4.1.2.2 Complex First-Order Schematics

Multiple circles can be connected to the same circle by different arrows to create complex schematics. In general, complex schematics *that do not involve process symbols* are essentially just conveniences; they simply enable one to reuse graphical elements and enable one to make

---

[18]That is, in terms of the elaboration language, Figure 4-11 translates to (conveys-to Conv-2 CB-J27-S121 PPV-1`

several assertions in the language by means of a single complex schematic. Thus, for instance, if one wished to express both that sparkplugs can be parts of engines and that engines can be parts of cars, there is no need for two circles representing the kind **engine**. Rather, the two facts in question can be expressed more succinctly, as in Figure 4-12.

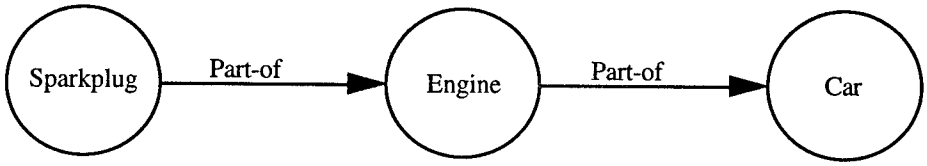


Figure 4-12. A Small Complex Schematic

Similarly, one might want to add the information that, in the given domain, cars can be made in Detroit and from there shipped to dealers, conveniently expressed as in Figure 4-13.

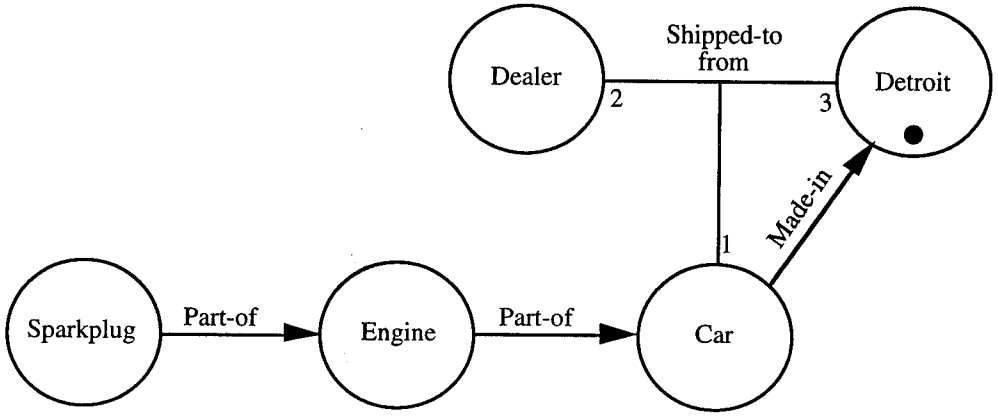


Figure 4-13. Complex Schematic Involving Multiple Relations

Complex schematics that do involve process symbols are discussed in Subsection 4.1.5.

4.1.2.2.1 Single Relation Complex Schematic Convention

At the same time, it will commonly be the case that an IDEF5 schematic may involve only one type of relation. In such cases, to prevent needless clutter, the modeler can omit labels and simply note the (single) meaning of the relation symbols at the bottom of the diagram, as illustrated in Figure 4-14:

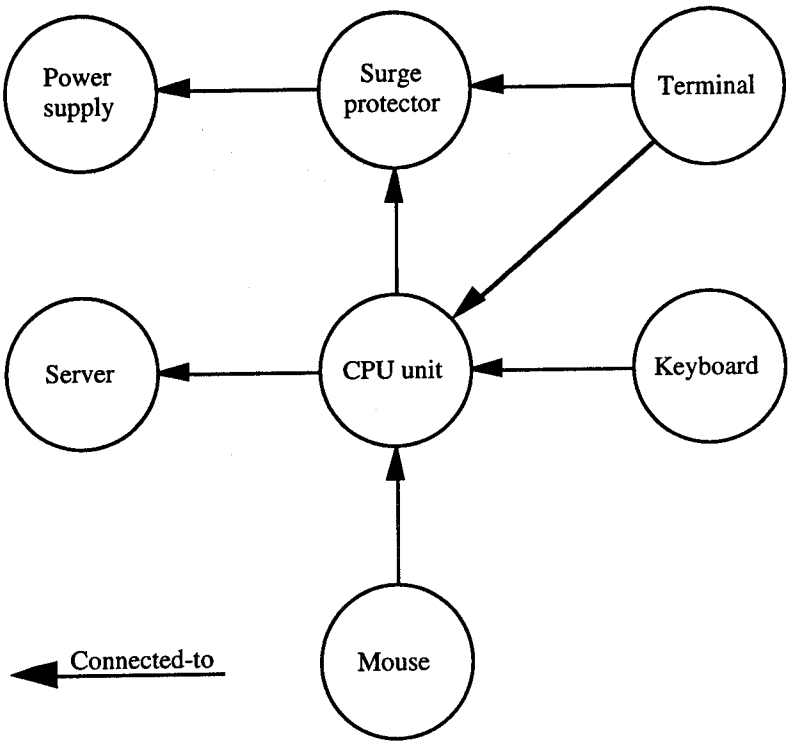


Figure 4-14. Peripheral Connections to a Personal Computer

Note also that the **connected-to** relation is included in the IDEF5 relation library (see Appendix A).

4.1.2.3 Second-Order Schematics

As noted, second-order relations are relations that hold between second-order objects (i.e., kinds and first-order relations) or between first-order and second-order objects. A paradigm of the former is the **subkind-of** relation between kinds, while a paradigm of the latter is the **instance-of** relation. A distinct type of arrow is needed to represent second-order relations because both types of arrows connect circles representing kinds. Because the associated semantics in the two cases are quite different, to avoid ambiguity, separate constructs must be used.

The basic form of a second-order schematic looks just like that of a first-order schematic, except for the presence of a second-order relation arrow instead of a first-order relation arrow.

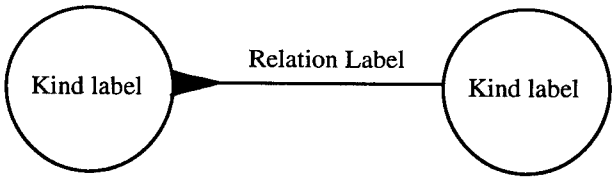


Figure 4-15. Basic Second-Order Schematic

The semantics for second-order schematics is much more definite than the semantics for first-order schematics. Specifically, these schematics are about the indicated kinds directly, rather than about their instances: Figure 4-15 means that the kind represented by the left-hand circle stands in the (second-order) relation, indicated by the arrow with the kind represented by the right-hand circle. Furthermore, note that the default semantics is not qualified; unlike first-order schematics, the semantics is not merely about how things can be in the domain but about how two kinds are in fact related. The reason for this is that such second-order assertions generally concern the *natures* of the kinds in question, and, thus, are not usually dependent on the contingencies of the domain though this is certainly not always so. Figure 4-16 illustrates a schematic involving the distinguished second-order relation **subkind-of**, which is provided as an IDEF5 primitive:

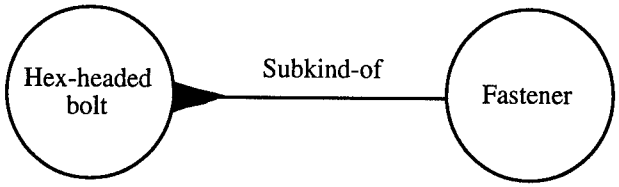


Figure 4-16. Example of a Second-Order Schematic with *Subkind-of*

By the semantics just given, the kind **hex-headed bolt** is a subkind of the kind **fastener**.[19] Similarly, the schematic in Figure 4-17 expresses that there are more U.S. citizens than Canadian (i.e., more literally, that the kind **U.S. Citizen** has more instances than the kind **Canadian Citizen**)

[19]In terms of the elaboration language again, we have simply (subkind-of hex-headed-bolt fastener).

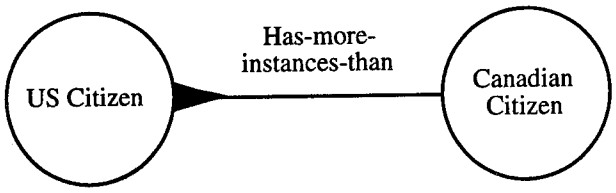


Figure 4-17. Example of a General Second-Order Schematic

Because the subkind relation is so common within ontologies, the default meaning of the second-order relation arrow with no associated label represents the subkind relation, thus permitting users to avoid having to attach the label **subkind-of** repeatedly throughout a schematic.

4.1.2.4 Relation Schematics

This section describes the use of IDEF5 schematics to capture and display relations between first-order relations. The main motivation for using schematics to capture and display such second-order relations is derived from the observation that people often describe and discover new concepts in terms of existing concepts. This is consistent with Ausubel's theory of learning, wherein learning often occurs by subsuming new information under more general, more inclusive concepts [Novak and Gowin 84, Sarris 92]. Using this hypothesis, a natural way to describe a new (or poorly understood) relation is to connect it to a relation that is already well understood and, more generally, to categorize its place in a "conceptual space" of other relations. The IDEF5 relation library (Appendix A) provides a baseline reference for users of IDEF5 to aid the discovery and characterization of relations using relation schematics.

The basic syntax of relation schematics is illustrated in Figure 4-18.

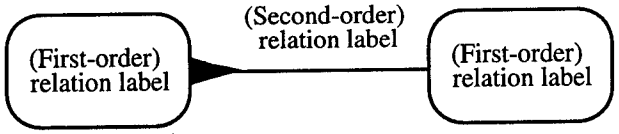


Figure 4-18. The General Form of a Basic Relation Schematic

It should be noted that the syntax of Figure 4-18 is identical to that of second-order schematics, except that first-order relation symbols are substituted for kind symbols. The use in this context of a second-order relation symbol is not mere notational parsimony, because kinds and first-order relations are of the same logical type. As noted in Section 2, kinds are properties of individuals, and first-order relations are associations between individuals, that is, properties of *n*-tuples. Thus, relations between first-order relations are of the same logical type as relations between kinds: both

relate entities that hold (or not) with respect to individuals and, hence, are second-order relations (as indicated explicitly in Figure 4-18 to avoid confusion).

4.1.2.4.1 Using Relation Schematics for Relation Analysis

The use of relation schematics to facilitate conceptual analysis involving relations (both relations between kinds and relations between relations) will now be illustrated. Consider the ontology for an engineering Bill Of Materials (BOM) of an automobile manufacturing company. The Part-of relation plays an important structural role in the BOM. The data acquired in the data collection process might contain the following statements about the BOM:

- An automatic transmission is a variant of the transmission.
- A manual transmission is a variant of the transmission.
- A radio is an option of the car.

A **Variant** of a product is an essential characteristic of the product that is often determined by customer choice [Anupindi 92]. The customer picks one of several possible variants. In this example, automobile customers choose between automatic and manual transmissions. Notice that **having a transmission system** is an essential property of the car (for most contemporary auto makers). The choice of a particular variety of transmission is a choice exercised by the customer.

An **Option** is a feature of a product that the customer chooses. Options are different from variants in that options can be completely eliminated from a product, whereas variants are required characteristics. In this example, the radio is not essential to the function of the car and can be optionally excluded.

Based on an analysis of the source statements, the IDEF5 developer hypothesizes the existence of two relations, and selects the relation names **Variant-of** and **Option-of.** These relations are mapped graphically on a relation schematic in the manner shown in Figure 4-19.

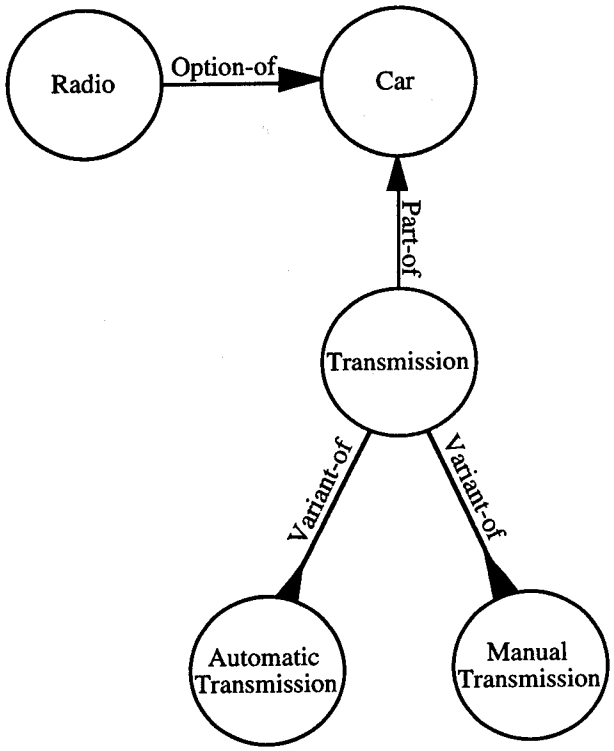


Figure 4-19. Bill of Material Relation Schematic

At this stage, an IDEF5 developer would browse the IDEF5 Relation Library. He or she may notice that the **Variant-of** and the **Option-of** relations are conceptually similar to some of the library relations. Specifically, he or she may realize that the **Option-of** relation is a specialization of the **Part-of** relation.

Suppose that these insights are now represented by the second-order relation **Specialization-of** in the relation schematics shown in Figure 4-20.

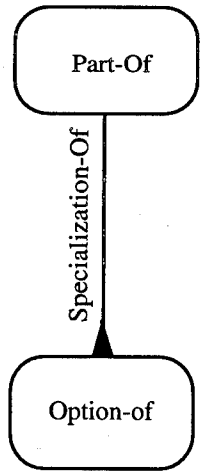


Figure 4-20. Relation Schematics Involving the *Specialization-of* Relation

The higher-order **Specialization-of** relation is used to assert the generalization-specialization relation between the indicated first-order relations. Further reflection leads the IDEF5 developers to draw a more detailed relation schematic that places the **Option-of** relation in a relation taxonomy diagram (i.e., a special type of relation schematic that shows a hierarchy of relations that are associated by generalization-specialization relationships). The taxonomy in question is exhibited in Figure 4-21.[20]

[20]A more complete taxonomy diagram of the meronymic (part-of) relations is given in the IDEF5 Relation Library (Appendix A).

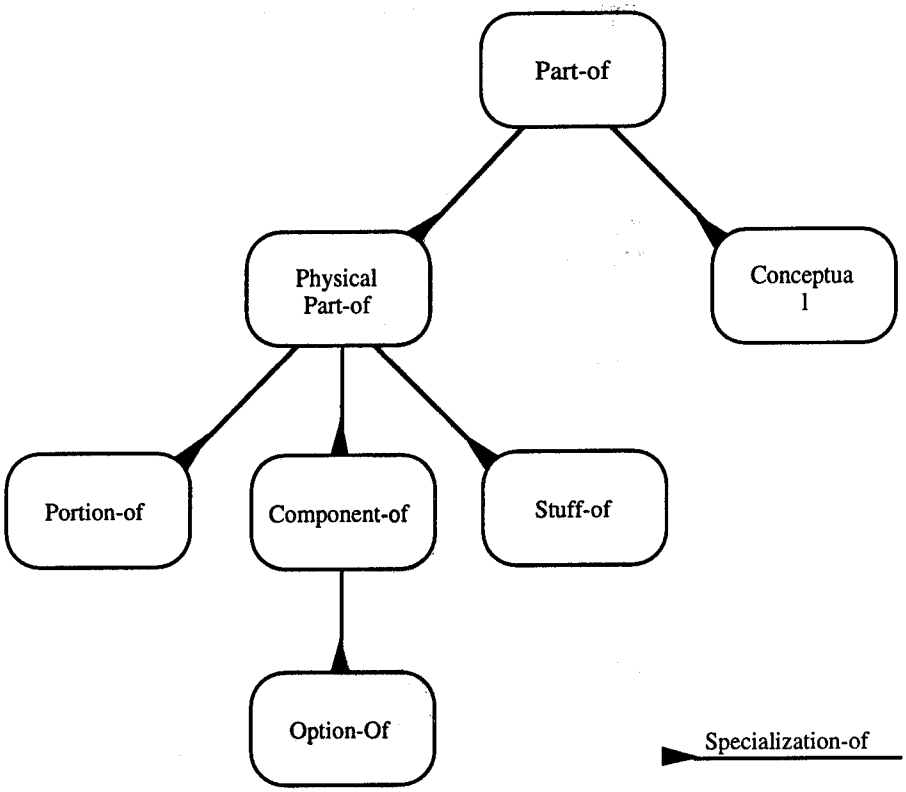


Figure 4-21. A Partial Relation Taxonomy of the Part-of Relation

It is to be noted that the single relation complex schematic convention is operative with respect to second-order relations in Figure 4-21, and also that complex second-order schematics can be constructed in the same manner as first-order schematics. That is, one can "reuse" kind and (first-order) relation symbols within a single schematic. For instance, the information contained in Figure 4-19 and Figure 4-20 could be expressed in the single schematic in Figure 4-22.

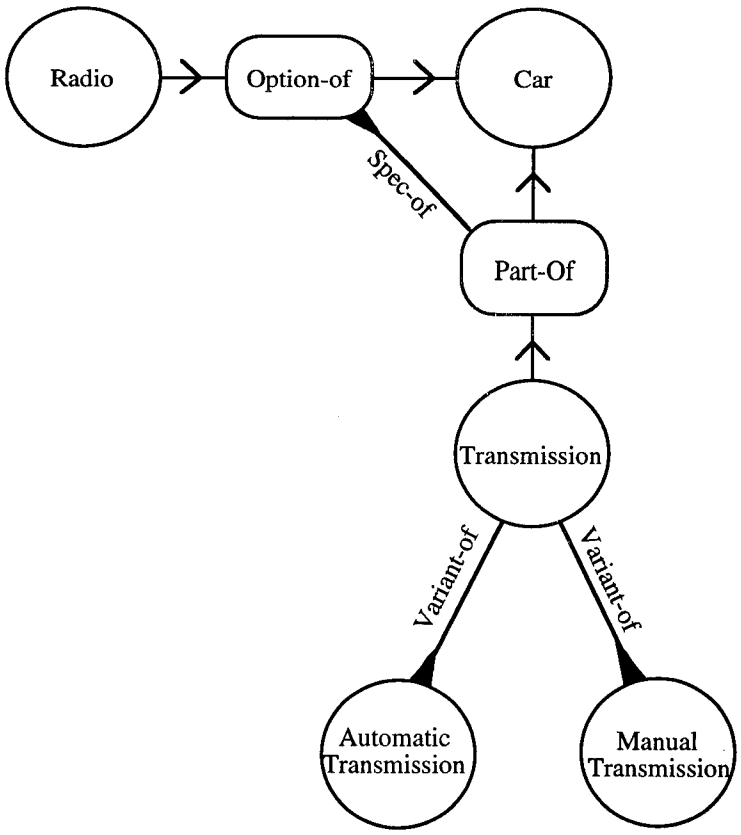


Figure 4-22. Complex Second-Order Relation Schematic

4.1.2.5 Referents

An IDEF5 referent is a modeling artifact used to indicate a concept in any another IDEF method. Referent rectangles have the following information items.

- **Referenced Concept Label** This is the label of the concept (within an IDEF model) that is being referenced. For example, an activity kind may reference an activity label in an IDEFØ model.
- **ID** This is the identification label for the Referent.
- **Method Name** This is the name of the IDEF method that is being referenced.

IDEF5 Referents provide a mechanism to link IDEF5 with other IDEF methods. For example, a kind in IDEF5 may be represented as an Entity in IDEF1, and the IDEF5 developer can make this reference explicit within the IDEF5 model.

4.1.3 Composition Schematics

Because the **part-of** relation is so common in design, engineering, and manufacturing ontologies, the "part-of" label, and associated axioms, are included in the IDEF5 languages. In particular, this capability enables users to express facts about the composition of a given kind of object. In general, this is achieved by means of schematics of the form illustrated in Figure 4-23.

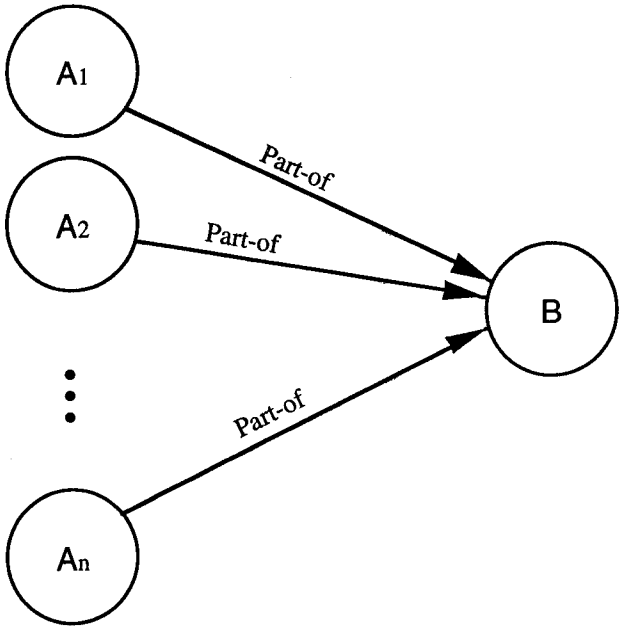


Figure 4-23. Composition Schematic

As noted in Section 4.1.2, Figure 4-23 is simply a convenient way to draw multiple instances of the basic first-order relation schematic illustrated in Figure 4-4, and hence, the default semantics of Figure 4-23, means that $\mathbf{A_1}$'s (i.e., instances of $\mathbf{A_1}$) can be parts of **B**'s, $\mathbf{A_2}$'s can be parts of **B**'s, . . ., and $\mathbf{A_i}$'s can be parts of **B**'s. However, in the context of **part-of**, it is frequently the case that a stronger reading is desired. For instance, in a bill of materials, one wishes to say not simply that $\mathbf{A_1}$'s can be parts of **B**'s, and so on, but that every **B** *does* in fact consist of an $\mathbf{A_1}$, an $\mathbf{A_2}$, and so forth. For example, one might wish to represent the component structure for a certain kind of ballpoint pen, as in Figure 4-24.

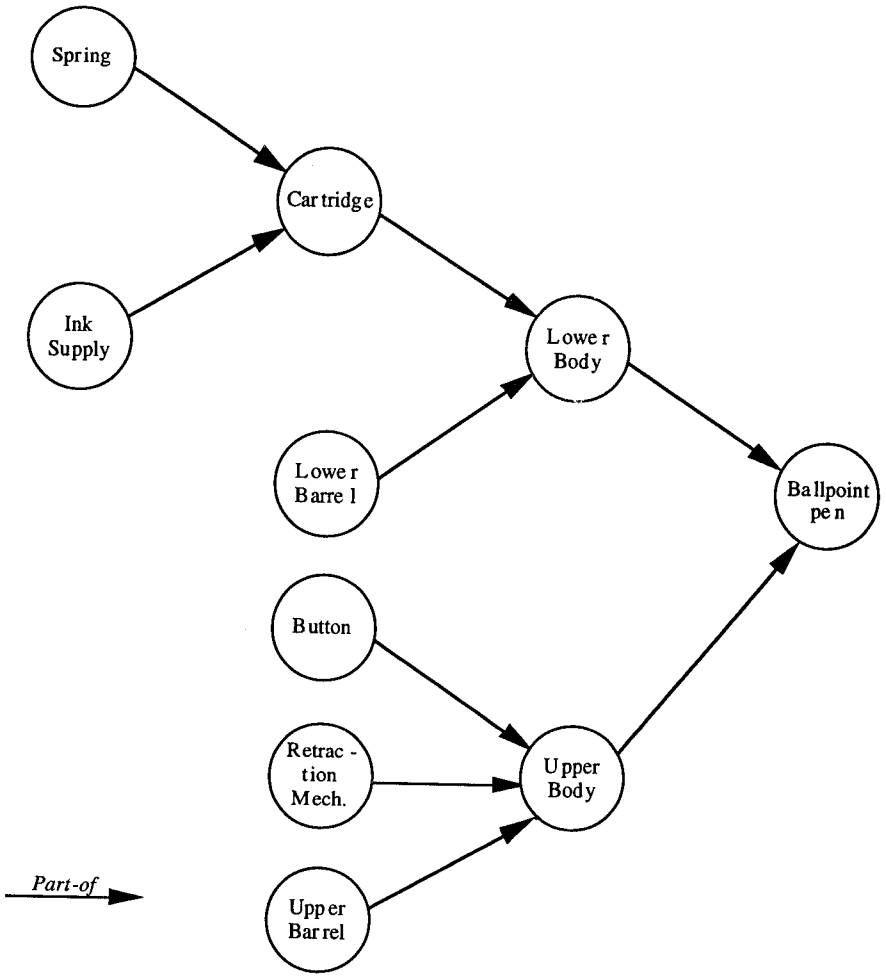


Figure 4-24. Composition Schematic for the Kind Ballpoint Pen

To capture this stronger meaning one must resort to the Elaboration Language (Subsection 4.2) and, for each instance of Figure 4-23, add the statement that every **B** does in fact have an $\mathbf{A_1}$, and $\mathbf{A_2}$, . . ., and an $\mathbf{A_n}$ as parts.[21] On this stronger semantics, then, the schematic in Figure 4-24 expresses that a ballpoint pen in the domain in question has both an upper body and a lower body, that the former consists of a button, a retraction mechanism, and an upper barrel, while the latter consists of a lower barrel and a cartridge, which in turn consists of a spring and an ink supply.

[21]Specifically, in the case of a kind B whose instances have three parts of kinds A1, A2, and A3, one would add the elaboration language statement (forall ?x (-> (B ?x) (exists (?y1 ?y2 ?y3)(and (A1 ?y1) (A2 ?y2) (A3 ?y3) (part-of ?y1 x) (part-of ?y2 x) (part-of ?y3 x)))).

4.1.3.1 Hiding Composition Information

As Figure 4-24 illustrates, composition schematics can be quite detailed. Such detail can cause a great deal of clutter in an IDEF5 diagram. For instance, in addition to describing the component structure of the kind **ballpoint pen**, one might also want to talk about many of the other relations it and its instances are involved in, for example, that the pens can be made in Sequim, Washington, that fountain pens generally cost more than ballpoint pens, that **ballpoint pen** is a subkind of **pen**, and so on. Hence, in many contexts, the component structure of the kind might well be irrelevant, and in such cases it would be useful to be able to hide that information. That such information is being hidden is indicated on a diagram by using a double circle to represent the kind (instead of a standard single circle), along with an upper case 'P' (for **part-of**) in the top of the circle to distinguish the kind of information that is being hidden, as illustrated in Figure 4-25.

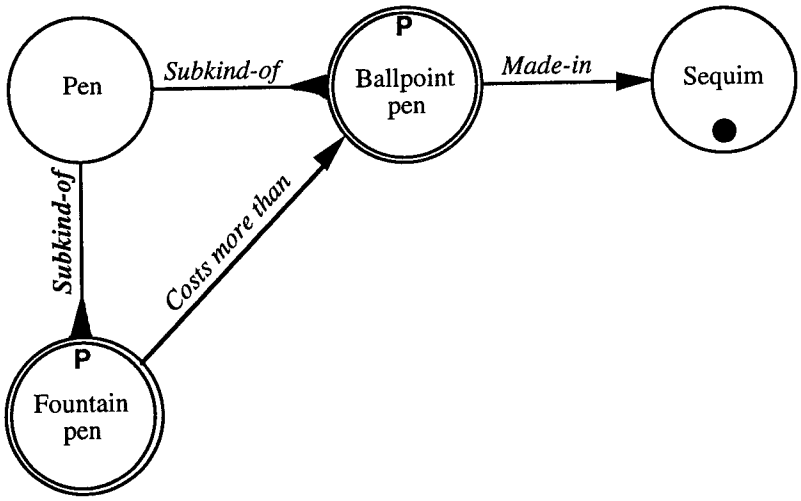


Figure 4-25. Hiding Composition Information

Note that this example illustrates the use of first- and second-order relation symbols in the same diagram.

4.1.4 Classification Schematics

Among the more commonly used structuring mechanisms used by humans to organize knowledge are taxonomy diagrams [Brachman 84]. Domain experts engaged in knowledge acquisition often make statements such as **A is a B**, **A is a type of B**, or **A is a kind of B**. The cognitive activity involved in organizing knowledge in this fashion is called *classification*. There are several identifiable varieties of classification. Two particularly prominent types of classification are *description subsumption* and *natural kind classification*. In description subsumption, (i) the defining properties of the "top-level" kind **K** in the classification, as well as those of all its

subkinds, constitute rigorous necessary and sufficient conditions for membership in those kinds, and (ii) the defining properties of all the subkinds are "subsumed" by the defining properties of **K** in the sense that the defining properties of each kind entail the defining properties of **K**; the defining properties of **K** constitute a more general concept.

In natural kind classification, by contrast, it is not assumed that there are rigorously identifiable necessary and sufficient conditions for membership in the top-level kind K, but that, nonetheless, there are some underlying structural properties of its instances that, when specialized in various ways, yield the subkinds of K. The best examples of such classification schemes are, of course, genuine natural kinds such as **metal**, **feline**, and so forth, but the idea can be extended to artifactual kinds like **automobile** and **NC machine**. These two types of classification are illustrated in Figure 4-26.

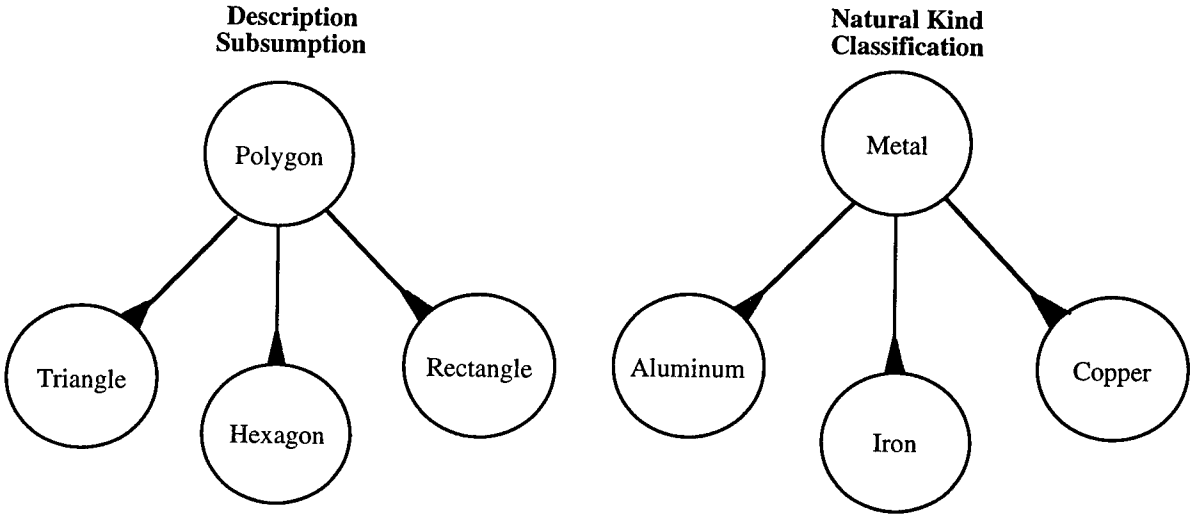


Figure 4-26. Different Types of Classification

Clearly, with its central notion of a kind, a natural application for the IDEF5 schematic language is the development of taxonomy diagrams, or as we shall call them, *classification schematics*.

Classification is typically much more detailed than the examples suggest. Most classification schemes will involve several levels of more specialized subkinds "below" more general kinds in the scheme. (Such schemes are often called 'is-a hierarchies,' but for the reasons adduced in Subsection 2.2.5, the use of 'is-a' is strongly discouraged in IDEF5; either the *subkind-of* relation or the *instance-of* relation should be used instead, depending on the intended meaning.) To illustrate, it is essential in project planning that one categorize the kinds of *resources* that will be needed for the project's success. Informally, a resource can be defined as an object that is consumed, used, or required to perform activities and tasks. Resources, therefore, play an

enabling role in processes. Classification diagrams provide a natural way of categorizing necessary resources, as, for example, in Figure 4-27. (Second-order relation symbols with no attached label, default to the subkind relation):

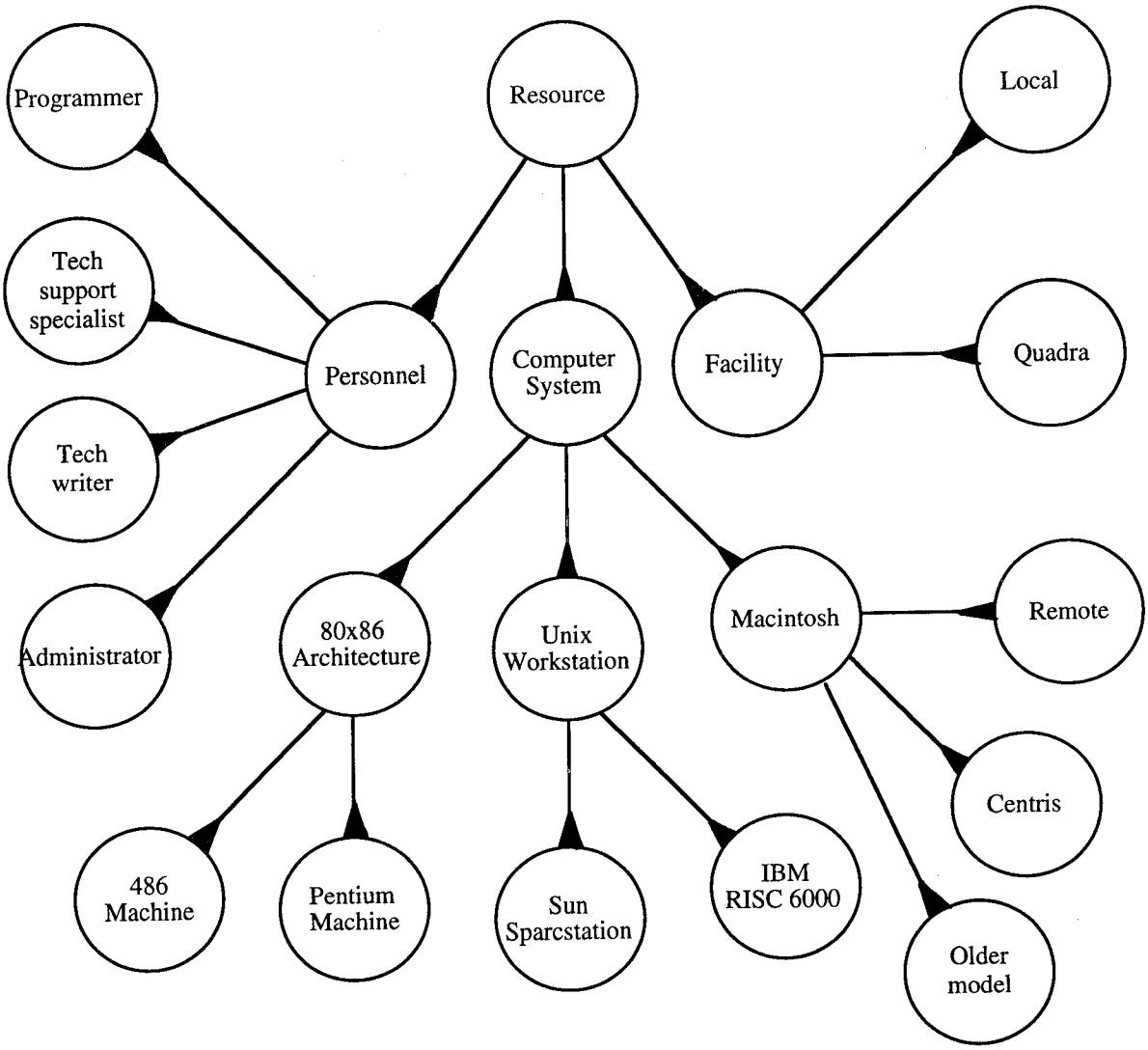


Figure 4-27. Classification of Resources

4.1.4.1 Hiding Classification Information

As with complex composition schematics, however, it often proves very useful to hide some detail in a classification schematic. Thus, in some contexts (e.g., those in which facilities and personnel need to be highlighted), information about computer systems might not need to be explicit. As with composition schematics, hidden information will be indicated by a double circle, annotated in this case with an upper case 'C' (for 'classification') at the top of the circle as shown in

Figure 4-28. Thus, one might hide that information and add information about facilities to obtain the following schematic.

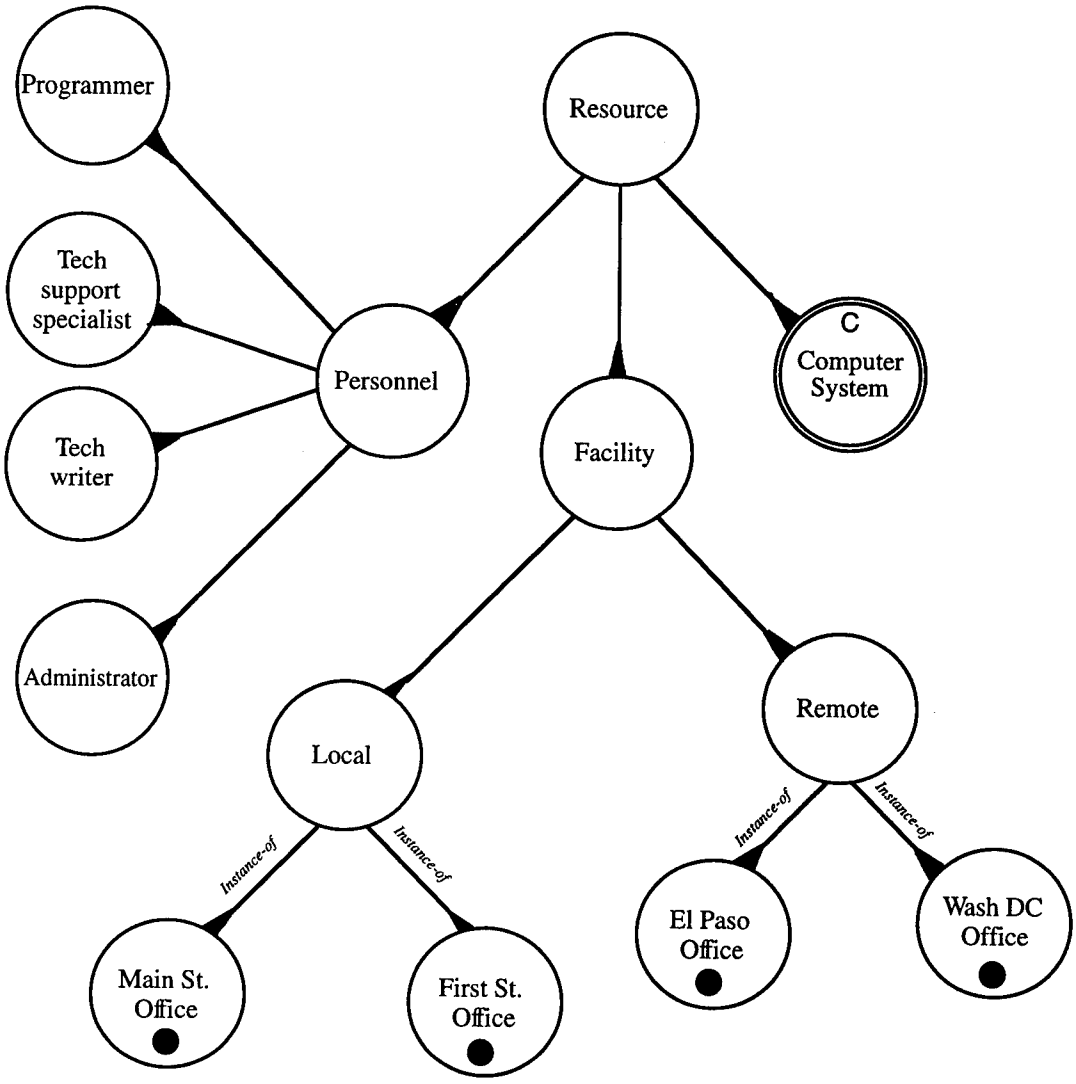


Figure 4-28. Classification of Resources with Hidden Information

4.1.5 Object State Schematics

As noted in Subsection 2.2.8, there is no clean division to be made between information about kinds and states and information about processes. This subsection describes how the IDEF5 schematic language enables modelers to express fairly detailed *object-centered* process information, that is, information about kinds of objects and the various states they can be in relative to certain processes. Diagrams built from these constructs are known as *object-state schematics* (OS's); the integration of OS's with the various kind schematics introduced above is discussed below.

In Section 2.2.8 two types of changes that can be observed in the objects undergoing processes were noted: change in kind and change in state. There is in fact no *formal* difference between these two types of change: objects of a given kind **K** that are in a certain state can simply be regarded as a constituting a subkind of **K**. For formal purposes, for example, **warm water** can just be regarded as a subkind of **water**. However, it is very useful to distinguish the two in the schematic language to indicate explicitly the kind of thing that is in a certain state. This is done by means of a colon notation, that is, **kind:state**. For example, warm water will be indicated by the label **water:warm**, frozen water by **water:frozen**, and so on. The notation is illustrated in Figure 4-29.

Figure 4-29. Kinds and States

In this context, the *subkind-of* relation is best thought of as a *state-of* relation, as illustrated in the classification diagram in Figure 4-30.

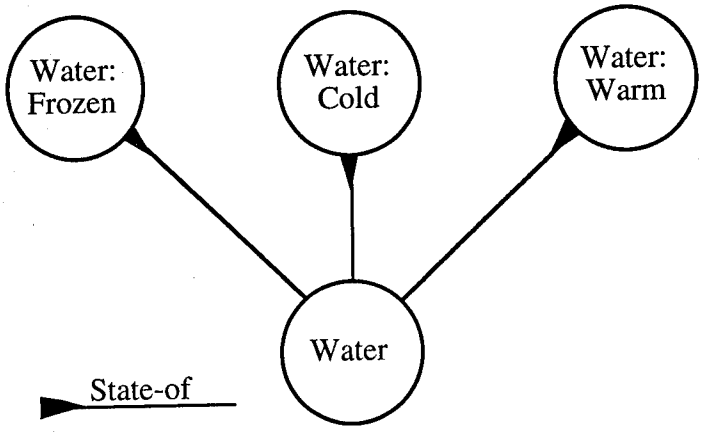


Figure 4-30. Schematic Depicting States of Water

To indicate how objects change either kind or state within processes requires an entirely new class of construct. The remainder of this section will be devoted to introducing the syntax and information semantics of these constructs.

4.1.5.1 Basic Object State Transition Schematics

The first and most basic construct is the simple *transition* link shown in Figure 4-31. Note that the presence of the open circle distinguishes an object state transition link from a general relation arrow.

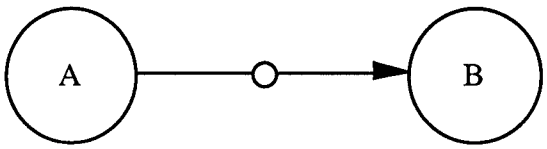


Figure 4-31. Basic State Transition Schematic

The circles labeled A and B in these links still indicate two kinds, and everything said about kinds and kind symbols in the previous subsection still applies. However, in characterizing in a general fashion the way objects are transformed through a process, it is natural to classify those objects in terms of the *states* they are in at various stages of the process. Hence, typically, the kind symbols in a state transition schematic will indicate a kind *in* a particular state, for example, **dry wood**, **warm water**, **unreworked part**, and so forth. To emphasize this role of the kind symbols, their meanings will often simply be referred to as *object states* or, simply, *states*.

Intuitively, an object state transition link indicates that there is an allowable transition such that an object in a given state **A** may be modified, transformed, or consumed so as to yield an object (possibly the same object) in a different state. Figure 4-31 depicts the situation where a certain type of transition from **A** to **B** is observed, but there is either no knowledge or desire to specify the process(es) involved in the transition.

It is important to note the distinction between the characterization of an object in a given state and the conditions or rules that govern how the object transitions to and from that state. In the IDEF3 process description capture method, conditions for entering and leaving a state are called *entry* and *exit* conditions, respectively. The IDEF5 Elaboration Language (Section 4.2) can be used to specify relevant entry and exit conditions for a given state.

Additional information about the process(es) involved in a state transition may be displayed in IDEF5, as shown in Figures 4-32 and 4-33.

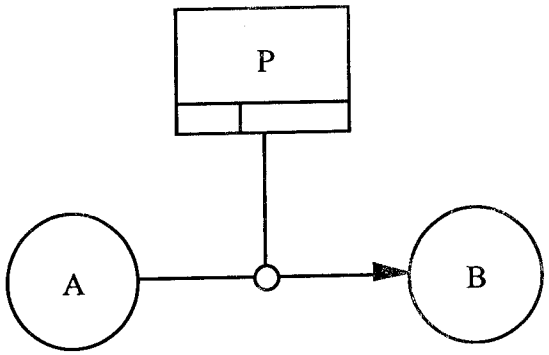


Figure 4-32. Schematic for Object State Transition within a Process

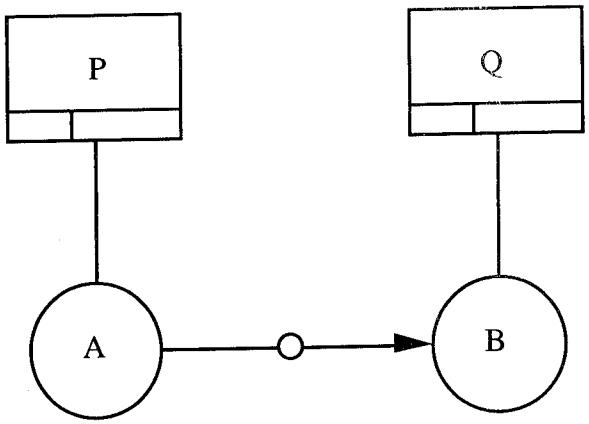


Figure 4-33. Schematic for Object State Transition between Processes

More precisely, the semantics of the state transition schematic displayed in Figure 4-32 is this: during some initial segment $\mathbf{t}_{init}$ of any occurrence **p** of the process **P**, there is an object **a** in state **A** (indicated by the atomic formula '**Aa**' in the "interval diagram" in Figure 4-34), and during some final segment t_{fin} (possibly the very last moment) of **p**, a possibly different object **b** is in state **B**.

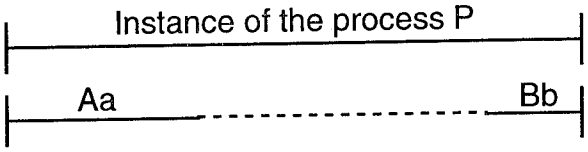


Figure 4-34. The General Semantics of Figure 4-32

It may happen, however, that an identified transition occurs not *within* a process, but between the end of a given process **P** and the start of some process **Q** (See Figure 4-33). A special case of this situation is discussed in section 4.1.5.3 wherein two contiguous processes **P** and **Q** meet. The transition arrow in such a schematic does not represent a single unspecified process but rather

marks the division in time between an object's being in state **A** at the end of some process **P** and the transformation to an object's being in state **B** at the start of another process **Q**, as displayed in the interval diagram in Figure 4-35. (The dashed line between the instances of **P** and **Q** indicates that they need not be temporally contiguous).

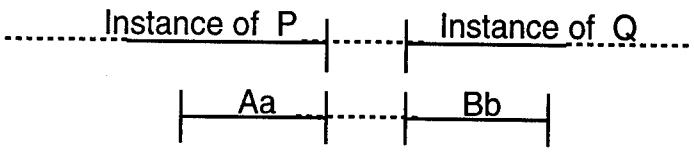


Figure 4-35. The General Semantics of Figure 4-33

4.1.5.2 Strong State Transition Schematics

The semantics of an object state transition link is open on the question of whether the object in state **A** at the beginning of a transition is *identical*[22] to the object in state **B** after the transition, as when an unpainted object becomes a painted object, or *distinct*, as when a piece of wood is transformed into a pile of ashes by a furnace. The basic state transition schematic should be used in any of the following three cases: (i) the objects at the beginning and end of the process are in fact distinct; (ii) it is not known whether or not they are distinct; or (iii) it does not matter one way or the other. On the other hand, if a modeler desires to represent explicitly that the object at the beginning of an instance of a state transition is identical with the object at the end, a *strong* state transition link should be used. This involves simply affixing a double tip on the arrow in a state transition link, as shown in Figure 3-36.

---

[22]Identity may be in terms of chemical structure, mass, physical form, function, etc. For example, grape juice becomes wine after undergoing a fermentation process. One might argue that the "stuff of" the kind **grape juice** is the same as that of the resulting kind **wine**. Other people having different attunements may perceive the two kinds as being entirely different based on, for example, chemical composition of the two kinds. It is therefore recommended that the assumed criteria for identity be established or characterized when there is possible ambiguity.

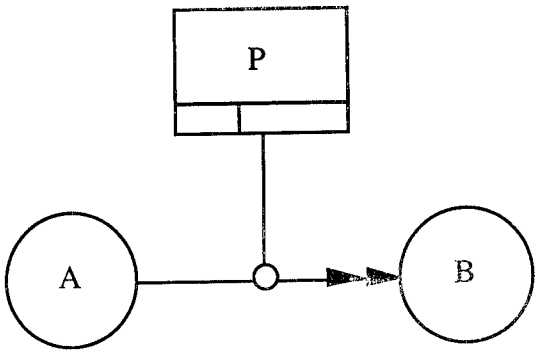


Figure 4-36. Strong State Transition Schematic

In order to allow for noninstantaneous state change within an instance **p** of a process, it is not generally required that **p**'s initial segment $\mathbf{t}_{init}$, during which an object is in state **A**, and its final segment $\mathbf{t}_{fin}$, during which an object is in state **B**, be contiguous or overlap. In such cases, the intervening period will, therefore, typically be thought of as a period during the process during which an object in state **A** is in the course of being transformed into something in state **B**, but during which there is actually nothing in either state, as in the period of time during which a quantity of water is heated from 5°C (state **A**) to 100° (state **B**). This indeterminacy in the semantics between instantaneous and noninstantaneous transitions is reflected graphically in the dashed line connecting the two states of the object **a** in Figure 4-37.

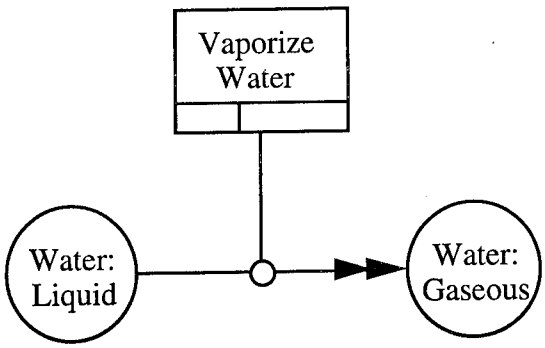


Figure 4-37. An Example of Strong State Transition Schematic

4.1.5.3 Instantaneous State Transition Schematics

As with the identity of objects in a transition, it is often useful to allow for the explicit representation of instantaneous state transitions within a process (relative to a certain temporal granularity). Accordingly, the OS syntax allows a modeler to tag the small circle in an object state transition link with a Δ, as in Figure 4-38, indicating thereby that the transition occurs over a period smaller than the smallest time unit Δ recognized in the context being modeled.

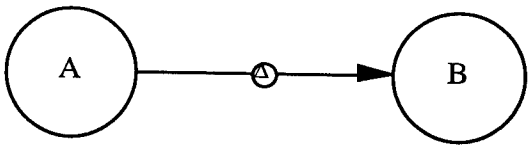


Figure 4-38. Instantaneous State Transition Schematic

A double arrow tip, of course, may be added if a strong transition is desired. For instance, when liquid oxygen is exposed to atmosphere, it transforms to gaseous state instantaneously (as shown in Figure 4-39).

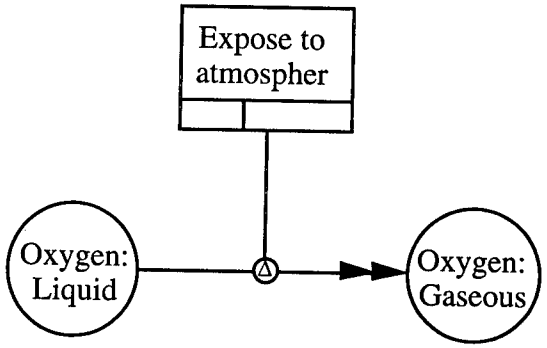


Figure 4-39. An Example of Instantaneous State Transition

One may be more interested in the processes which immediately precede and/or follow the transition rather than the instantaneous process which occurs upon transition. This situation is illustrated by the interval diagram in Figure 4-40.

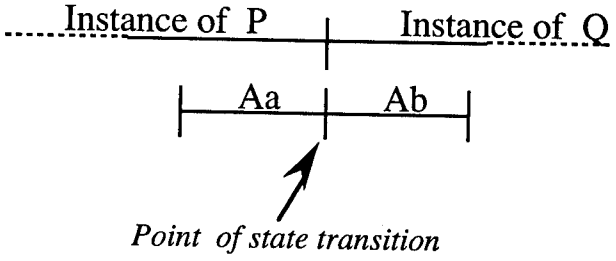


Figure 4-40. Interval Diagram for Figure 4-39

To express the content of Figure 4-40 precisely, the construct in Figure 4-41 is used. The transition arrow in such a schematic marks the division in time between an object's being in state **A** at the end of some process **P** and the transformation to an object's being in state **B** at the start of another process **Q**.

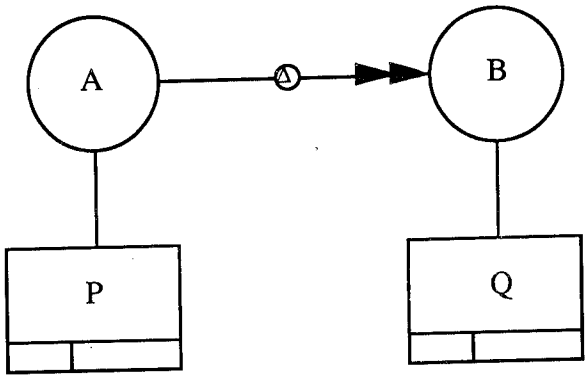


Figure 4-41. A Precise Expression of Figure 4-40

For example, one could imagine a cutting tool that is driven forward across a workpiece until it activates a limit switch whereupon the cutter is switched off and retracted to its starting position (Figure 4-42).

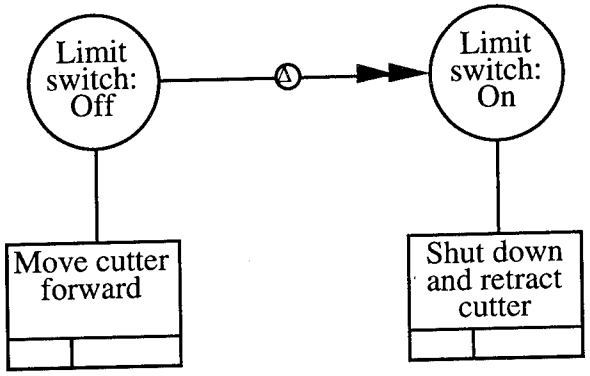


Figure 4-42. Cutoff Switch Example for Figure 4-41

4.1.5.4 Complex Object-State Schematics

As with the schematics introduced thus far, the simple object schematics introduced in the previous section can be combined to form more complex object-state schematics. Unlike complex schematics introduced to this point, however, complex object-state schematics are not simply notational conveniences; they carry additional meaning. For instance, Figure 4-43 expresses more than is pictured in Figure 4-41; the latter indicates nothing about the state of object **a** prior to its being in state **A** (in particular, nothing about state **D**), and nothing about the state of object **b** after its being in state **B** (in particular, nothing about state **C**).

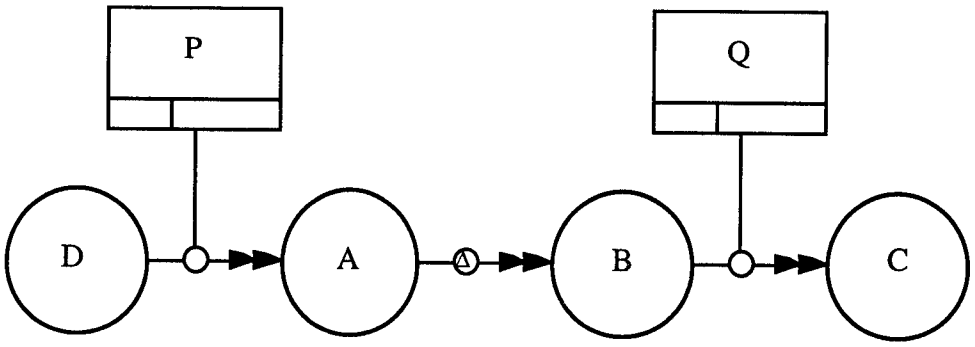


Figure 4-43. A More Informative Object State Transition Schematic

At times, one may only wish to record the bare transition itself, without providing any further detail about what happens on "either side" of the transition. In such a case, Figure 4-41 suffices. Otherwise, a diagram such as that in Figure 4-43 can be used. Th interval diagram for Figure 4-43 is shown in Figure 4-44. Hence, Figure 4-43 is appropriate, for example, in those cases in which one knows about, or wishes to express, the details of the process by which an object comes to be in state **A** and the details of what happens after that object is in state **B**.

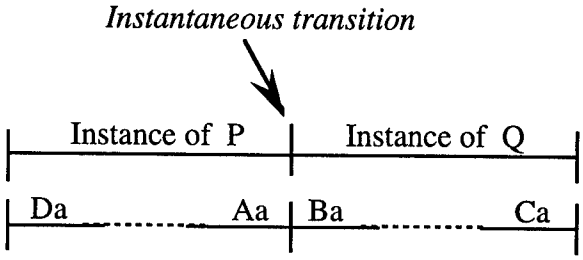


Figure 4-44. Interval Diagram for Figure 4-43

Another example of how complex schematics can provide additional information is as follows.

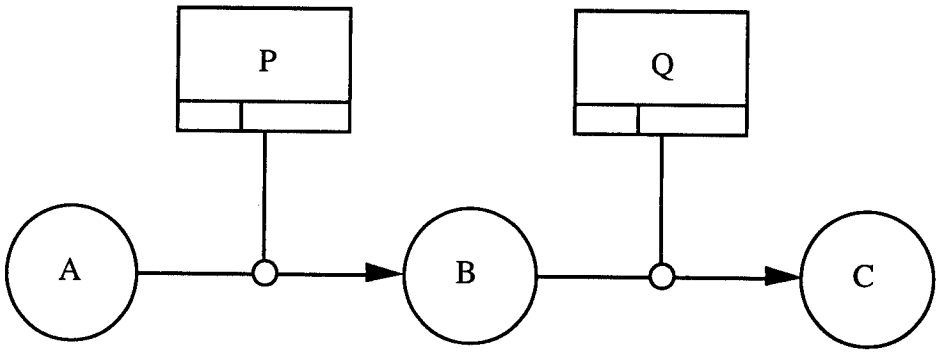


Figure 4-45. A Complex Object-state Schematic

Figure 4-45 means, first, that there is a transformation from something's being in state **A** to something's being in state **B** in instances of process **P** and, subsequently, a transformation involving *that same thing* in state B to something's being in state **C** in instances of process **Q**. That is, the fact that the symbols for object states **A** and **C** are linked to the symbol for **B** in Figure 4-45 indicates, first, that **P** and **Q** can be thought of as parts of a more complex process **R** that encompasses them both and, second, that the instances of **P** and **Q** within an instance of the overarching **R** *share* an object in state **B**. This general semantics is depicted in the interval diagram in Figure 4-46.[23]

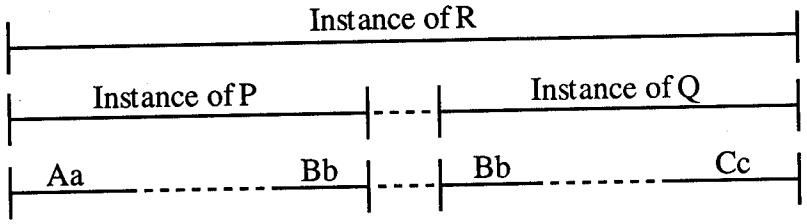


Figure 4-46. Default Semantics for Figure 4-45

(The semantics, of course, generalizes to more complex OS schematics involving more than two process symbols.) Figure 4-45 is to be contrasted with Figure 4-47 in which the symbols for states **A** and **C** are each linked to separate symbols for **B**.

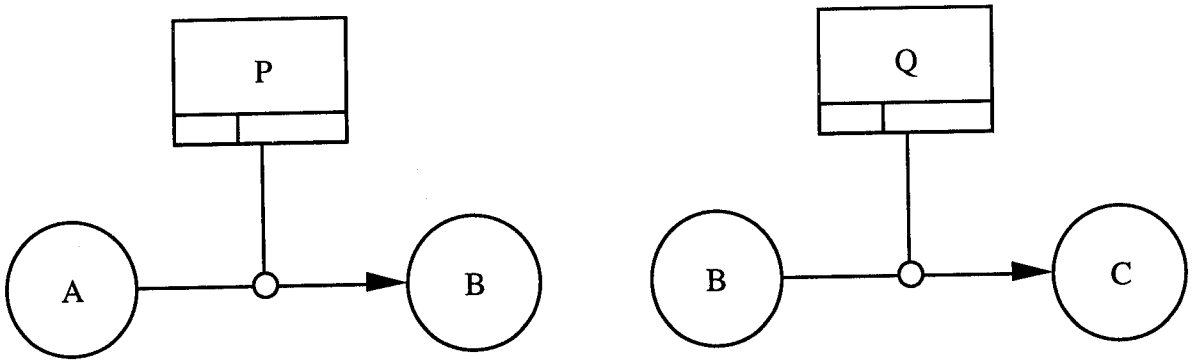


Figure 4-47. A Schematic Subsumed by Figure 4-45

---

[23]The second condition of the default semantics here might be too strong for some situations. For instance, a domain expert might want to represent a situation in which multiple parts may be produced through a given process **P** and inspected at random through some process **Q**. In general, then, in this case, although there is an overarching process **R**, there is no assumption that the object in state **B** at the end of (an instance of) **P** is identical with the object in state **B** at the beginning of **Q**. Such a situation can be represented, for example, by connecting the two symbols for **B** in Figure 4-47 with a weak transition arrow or, of course, can be expressed in the elaboration language directly.

Because the two schematics are not connected as in Figure 4-45, there is no implication that there is any overarching process **R** that subsumes both **P** and **Q**, nor is it implied that any instances of the two transitions share an object in state **B**. Figure 4-45 thus *implies* what is expressed by the two separate diagrams of Figure 4-47 but, in addition, implies more information.

4.1.5.5 State Classification Schematics

An important corollary of the inclusion of object states as possible meanings of kind symbols in transition schematics is that the *subkind-of* relation can, in these contexts, be thought of as the *state-of* relation, as illustrated in Figure 4-48.

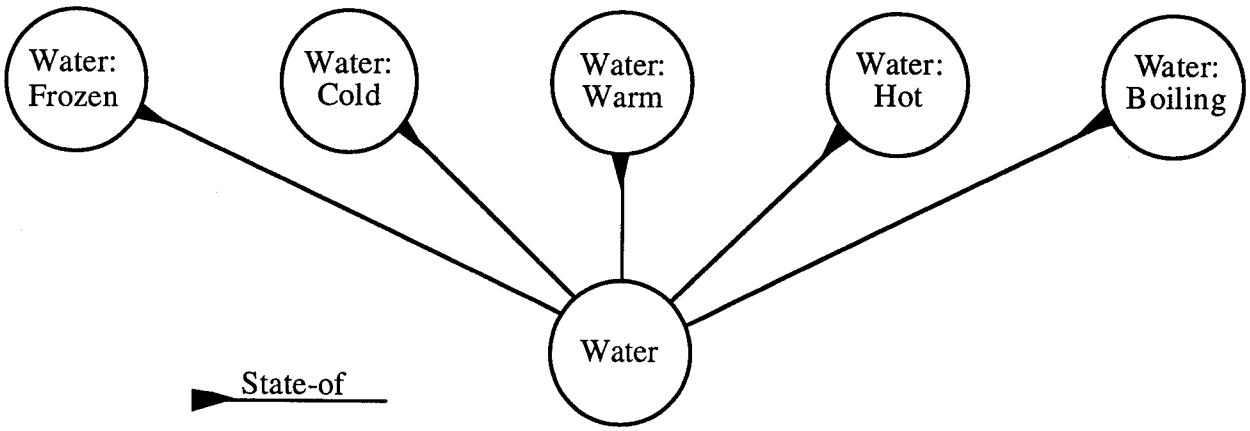


Figure 4-48. Schematic Depicting States of Water

Here, instead of various subkinds of the kind **water**, various possible *states* of water that can occur in the given domain are represented. Such schematics can be combined with standard OS schematics as in Figure 4-49.

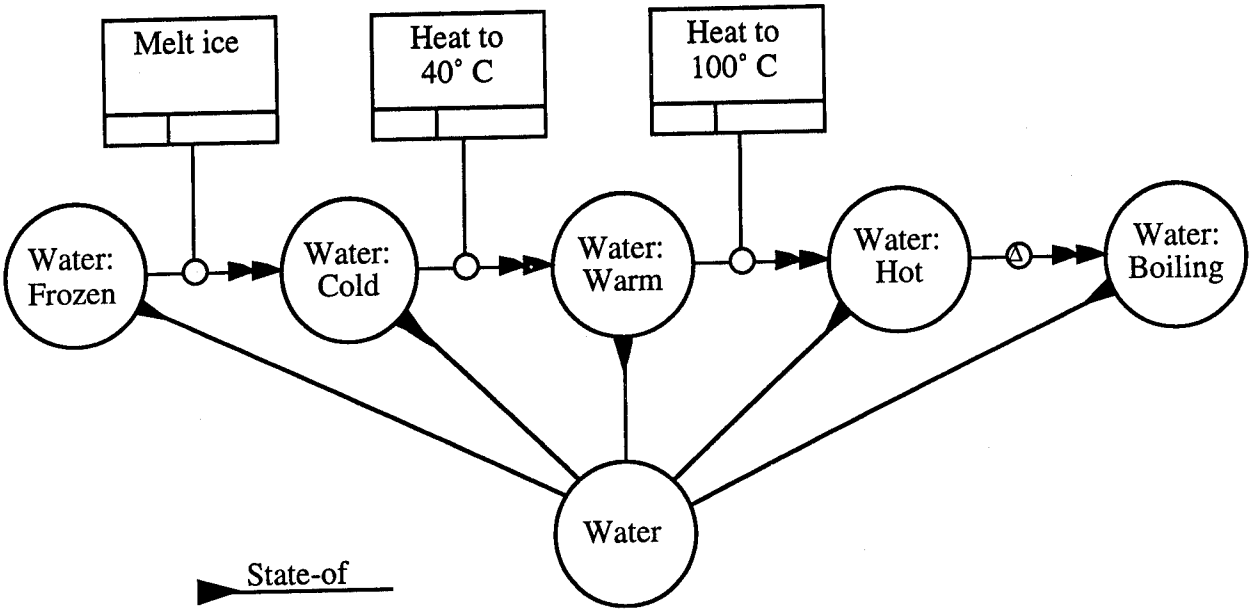


Figure 4-49. Combined Schematic Displaying States and State Transitions

4.1.5.6 State Composition Schematics

A particularly important point of contact between OS's and the basic IDEF5 schematics concerns compositions. The general OS composition schematic is illustrated in Figure 4-50.

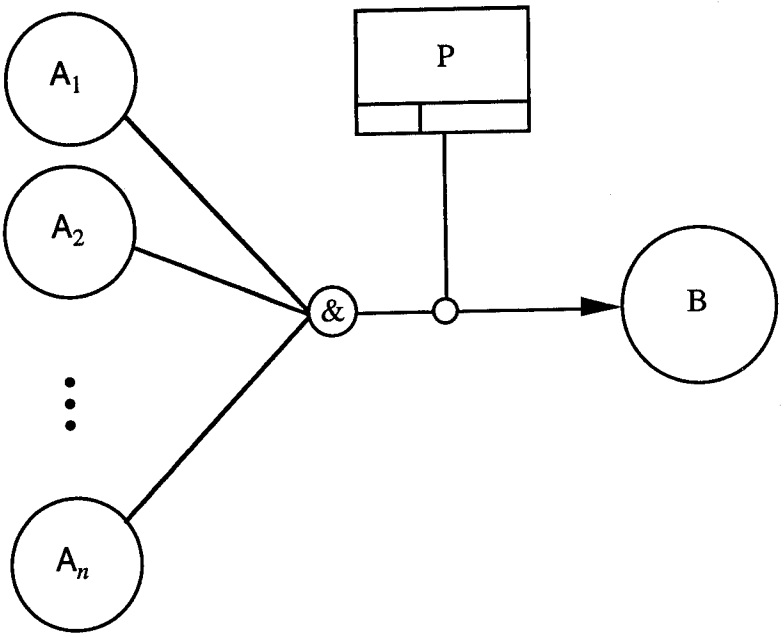


Figure 4-50. State Composition Schematic

The semantics for state composition schematics is a generalization of the semantics for state transition schematics. Intuitively, once again, the OS in Figure 4-49 represents a process type **P** in

which objects $\mathbf{a}_1, \ldots, \mathbf{a}_n$ in states $\mathbf{A}_1, \ldots, \mathbf{A}_n$, respectively, are involved in some process that yields an object **b** in state **B**. As with state transition schematics, each of the $\mathbf{a}_i$ should be in state $\mathbf{A}_i$ during some initial segment of **p**. It is not required that *every* $\mathbf{a}_i$ be in state $\mathbf{A}_i$ *throughout* some initial segment, as their coming into these states may be staggered throughout the course of **p**. It *is* required that every $\mathbf{a}_i$ be in state $\mathbf{A}_i$ prior to **b**'s being in **B** (or else, contrary to the intuitive meaning of the schematic, it would not be the case that the $\mathbf{a}_i$'s in the corresponding states are combined into **b**). This general semantics for Figure 4-50 is indicated graphically in Figure 4-51.

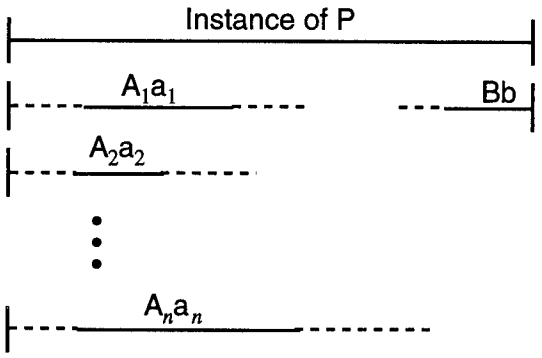


Figure 4-51. The General Semantics of a State Composition Schematic

The process represented in Figure 4-52 initially involves wood in a dry state and air that is oxygen rich. Objects in these states are then involved in the incineration process, which results in ashes. (Note that this example also illustrates that kind labels that don't involve reference to an object state can be used in OS schematics; the label 'Ashes', in particular, is not qualified in this manner. Such a usage arises in those situations in which the state of the object in a certain process is irrelevant from the perspective being modeled.)

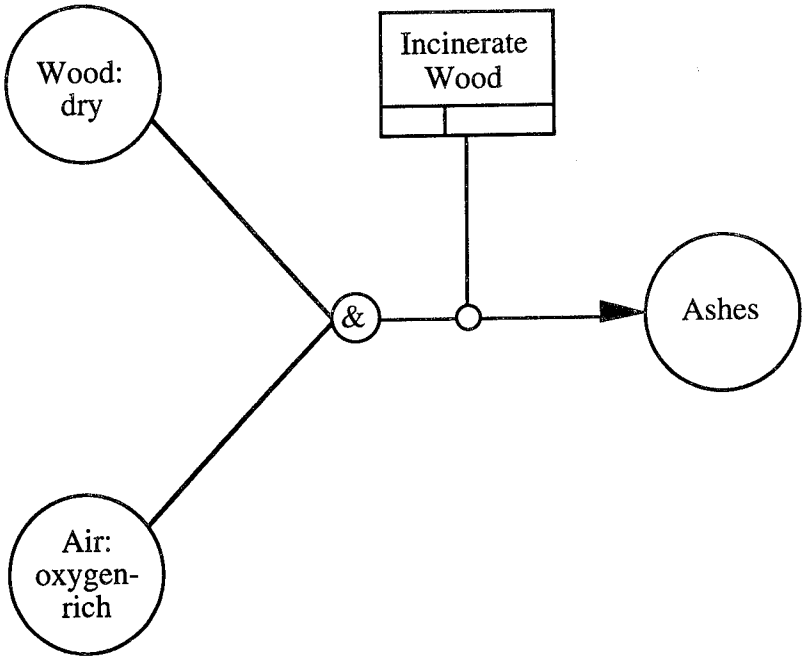


Figure 4-52. An Example of State Composition Schematic

Combining this semantics with the composition schematics of Figure 4-23 generates the notion of a *strict* state composition schematic, whose form is the same as a composition schematic, but with the label 'Part-of' attached to the arrow, as shown in Figure 4-53.

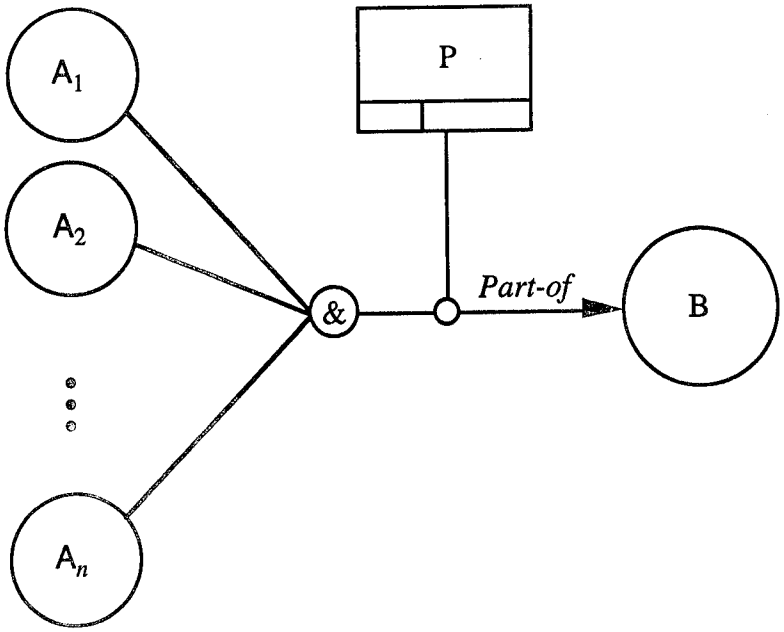


Figure 4-53. Strict State Composition Schematic

The semantics of a strict composition schematic is that, not only are instances $\mathbf{a}_1, \ldots, \mathbf{a}_n$ of $\mathbf{A}_1, \ldots, \mathbf{A}_n$, respectively, involved in a process that yields an instance **b** of **B**, but also that those

objects are all *parts* of **b**. This idea is illustrated in Figure 4-54 by explicitly adding process information to the complex composition schematic of Figure 4-23 depicting the component structure of a kind of ballpoint pen. [It should be emphasized that this schematic is not intended to represent the structure of all possible ballpoint pens (some ballpoint pens don't have a retraction mechanism, for example), but rather the *particular* kind of pen found in some hypothesized domain.]

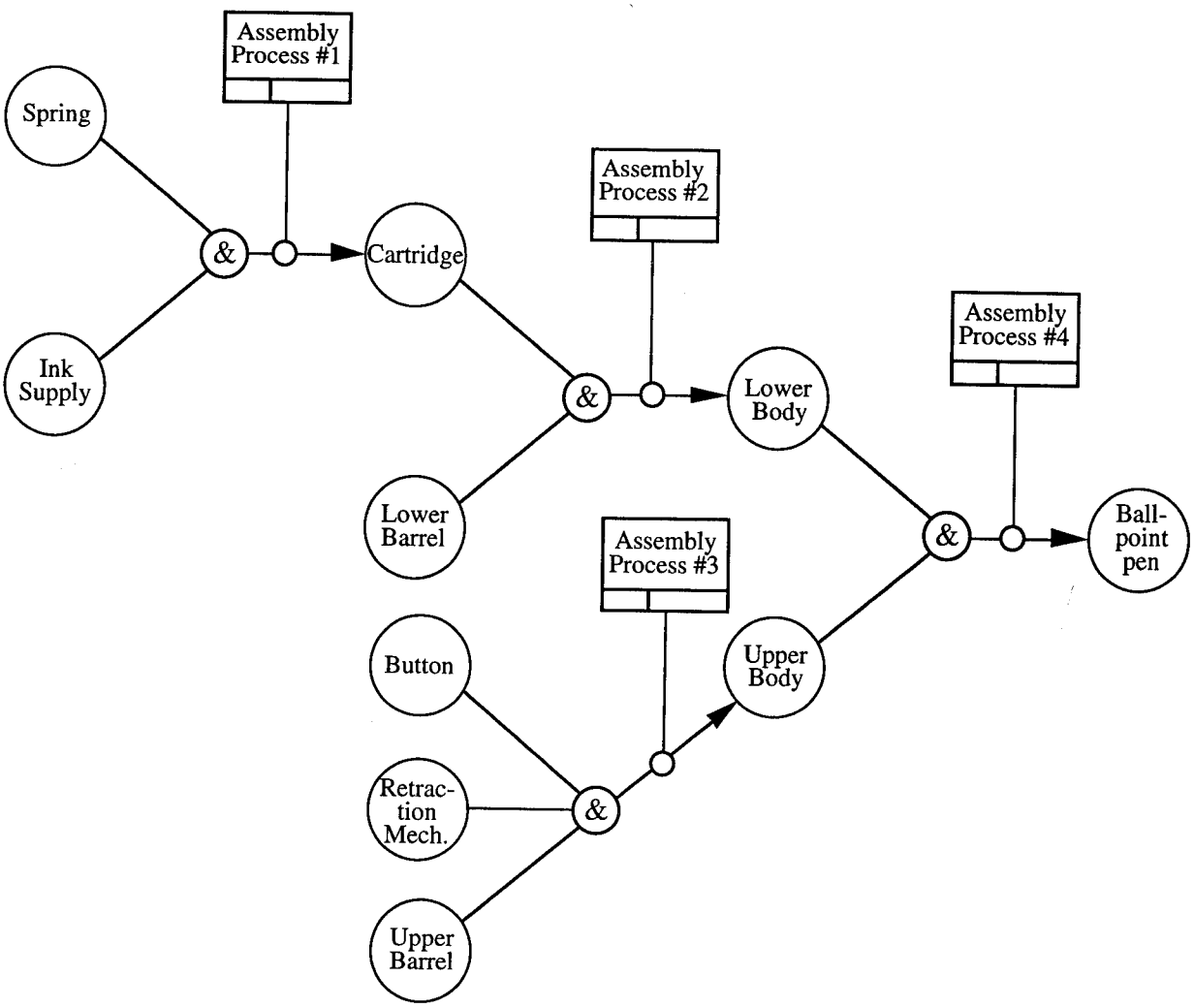


Figure 4-54. Complex Strict State Composition Schematic for the Kind Ballpoint Pen

In the usual case, the object **b** resulting from a composition process will be distinct from the objects $\mathbf{a}_1, \ldots, \mathbf{a}_n$ of which it is composed. However, the conception of objects here is flexible enough that this is not always so; a car body without a side-view mirror is intuitively the same as the car-body that results from affixing such a mirror. Hence, the representation of strong transition in composition schematics is permitted as well, indicated once again by means of double-tip

arrows, as in Figure 4-55. Note also that instantaneous transitions can also be represented by appropriate placement of the Δ symbol.

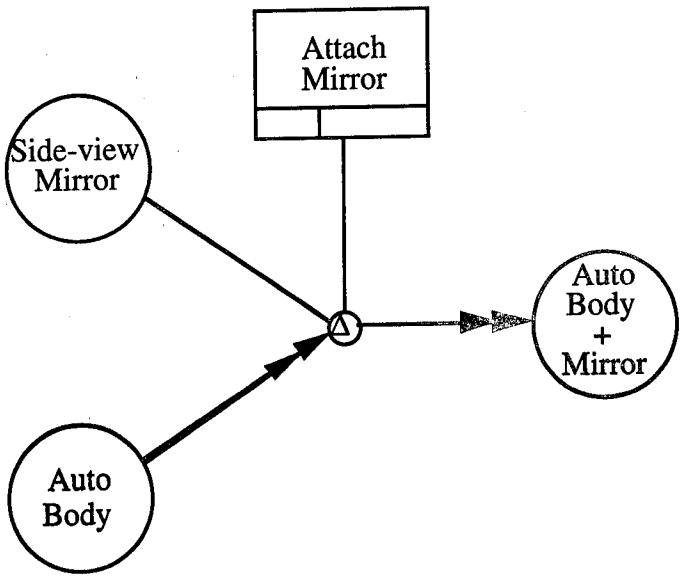


Figure 4-55. Strong State Transition in a Composition Schematic

In the context of OS's, where there is an explicit temporal component, the notion of object state *decomposition* makes sense. It is, furthermore, easy to specify, as it is just the inverse of state composition; its representation, shown in Figure 4-56, reflects this:

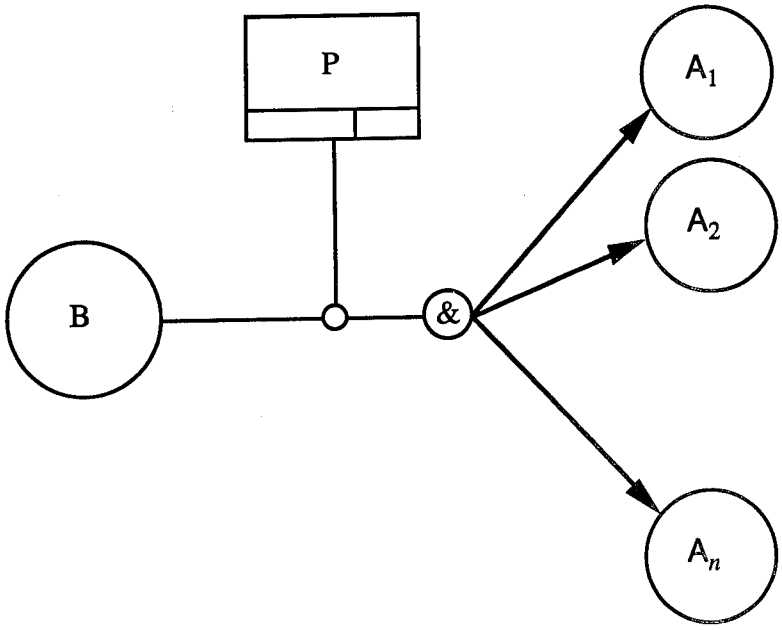


Figure 4-56. Object State Decomposition Schematic

The semantics of a state decomposition schematic as well is just the inverse of the semantics for composition schematics. And once again, double-tip arrows are permitted for indicating strong transitions. State transition schematics can be taken to be limiting cases of composition and decomposition schematics for the case $n = 1$.

The next OS construct to be introduced – a *disjunctive* state transition schematic – is a logical schematic that indicates that an object state may transition alternatively to one of a number of other states, as shown in Figure 4-57.

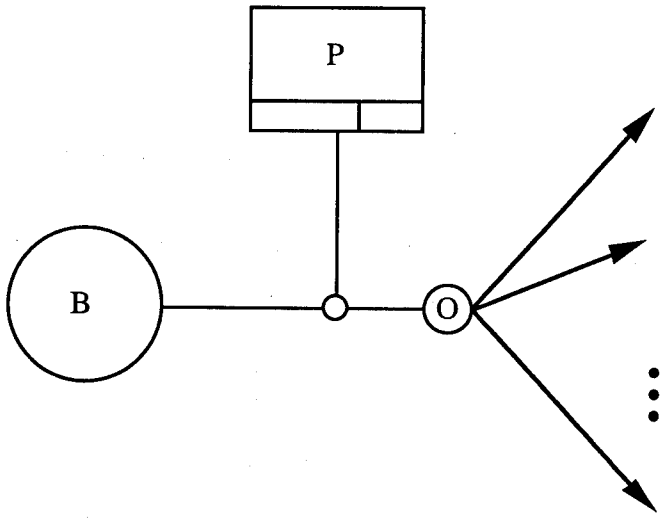


Figure 4-57. Disjunctive State Transition Schematic

The type of disjunction here is *inclusive*, in that it permits a transition from **B** to any of the subsequent states, possibly more than one. To indicate *exclusive* disjunction, which permits transition to no more than one of the subsequent states, the construct in Figure 4-58 is used:

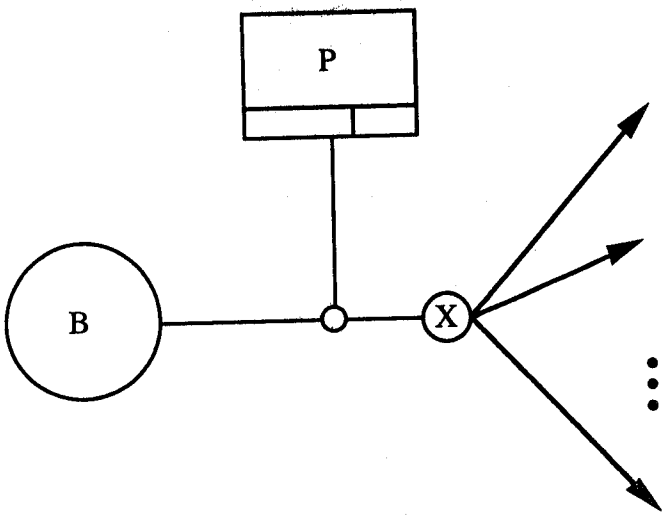


Figure 4-58. Exclusive Disjunctive State Transition Schematic

By the same token, a *conjunctive* schematic is introduced to indicate a transition from a given state to *all* of several subsequent states, as illustrated in Figure 4-59.

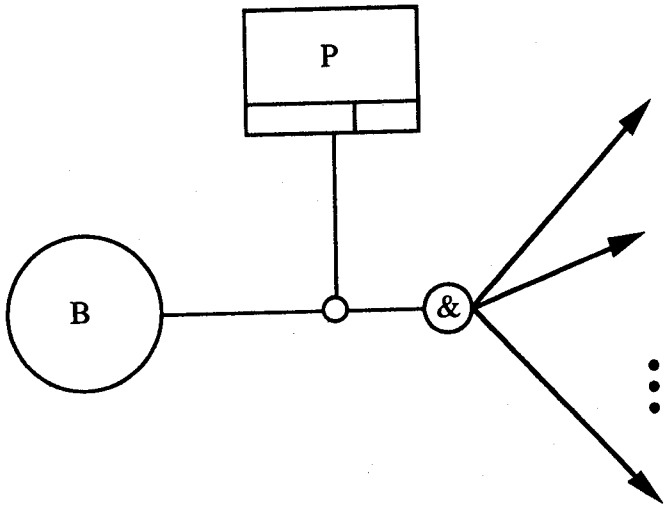


Figure 4-59. Conjunctive State Transition Schematic

In each case, the modeler is permitted to attach a process symbol to the junction or to each arrow extending from the junction (but not both). In the case of disjunctions, attaching a process symbol to the junction means that the indicated process can bring about any of the subsequent states. In the case of conjunctions, attaching a process symbol to the junction means that the indicated process brings about all of the subsequent transitions. (Decomposition, therefore, is just a special case of a conjunctive state transition schematic when the lines leading out from the junction indicate the **part-of** relation.) Judicious use of the Δ operator once again can be used to indicate which, if

any, of the transitions in an (inclusive or exclusive) disjunctive or conjunctive schematic are instantaneous.

As with composition, these logical schematics have "converses." Specifically, where the symbol '*' is O, X, or &, the diagram in Figure 4-60 is also a schematic:

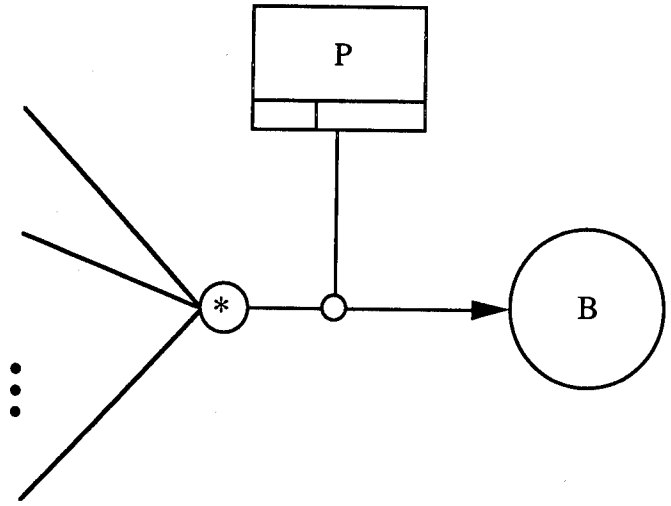


Figure 4-60. Converse Logical State Transition Schematic

The semantics in each case will be exactly the converse of the corresponding schematic above. (Hence, in particular, composition schematics will be special cases of the converse conjunction schematic.) It is conceivable that one object state might transition alternatively in many different ways, as illustrated in Figure 4-61:

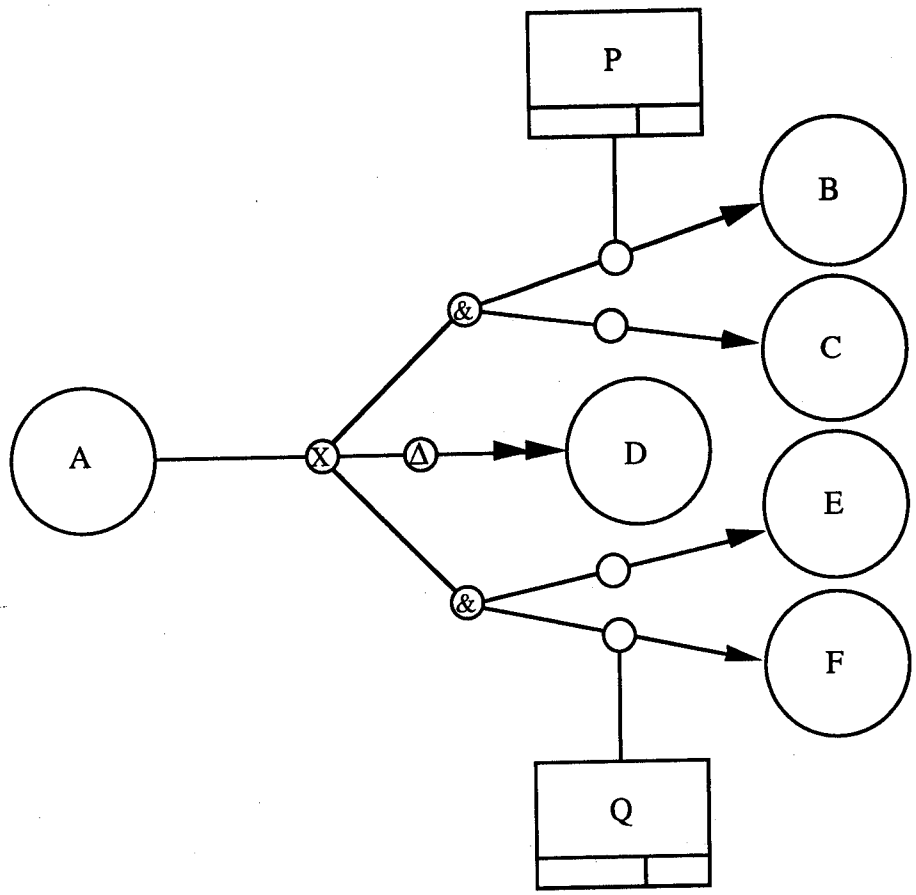


Figure 4-61. An OS Illustrating Possible Complex State Transition Logic

4.1.5.7 Hiding Object State Information

As with composition and classification schematics, it is possible to hide information in object state schematics. That is, for certain purposes, it may often prove useful to collapse complex state transition information about a given object into a single object state. For example, a series of state transitions involved in the process of heating water from freezing to boiling is depicted in Figure 4-62:

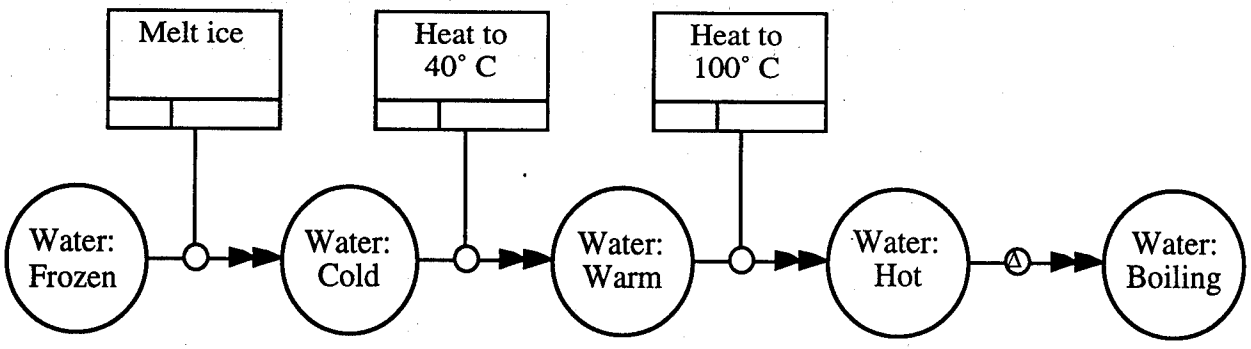


Figure 4-62. State Transitions in a Heating Process

If, from a certain perspective, the intermediate transitions from ice to boiling water are irrelevant, then these transitions can be hidden in a single state in which the only relevant state is the coarse-grained **Water being heated** as depicted in Figure 4-63; again a double circle is used, only in this case an 'S' indicates that the type of information that is hidden is state transition information:

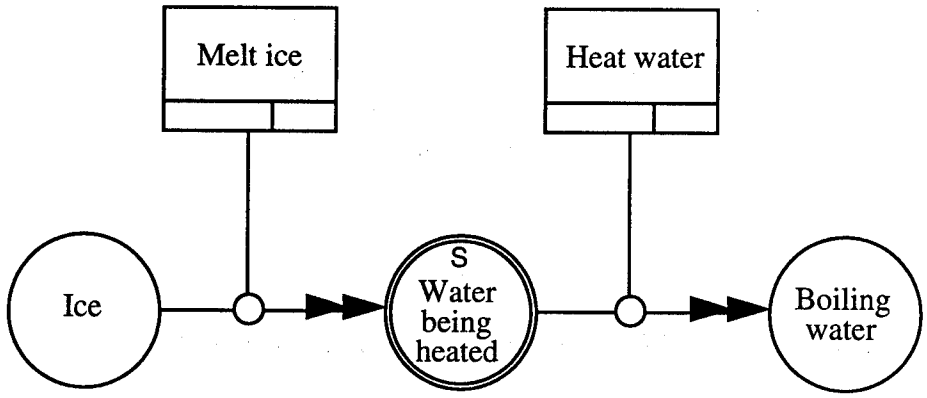


Figure 4-63. Hiding State Transition Information

The procedure for generating a coarse-grained schematic from a finer-grained schematic is not *quite* algorithmic. In the example, the kind symbol for **Water being heated** can be thought of as directly replacing the "schematic" of Figure 4-62, consisting of the middle three kind symbols and their connecting links. However, the instantaneous transition schematic in Figure 4-62 had to be replaced by an ordinary state transition schematic, and an appropriate label had to be found for the attached process box. The exact nature of this alteration had to be determined by the nature of the represented process and, hence, is, in general, a nonalgorithmic modeling decision.

4.1.5.8 Integrating Transition Schematics and Relation Schematics

OS's integrate naturally into ordinary IDEF5 schematics as the OS constructs are simple *extensions* of the schematic language presented above; one can supplement any schematic built from the

original constructs with OS constructs as desired. An example of this was seen in Figure 4-53, in which composition information and process information are integrated in a single generalized state composition schematic. Another simple example of a schematic that integrates both the original constructs and the OS constructs is seen in Figure 4-64.

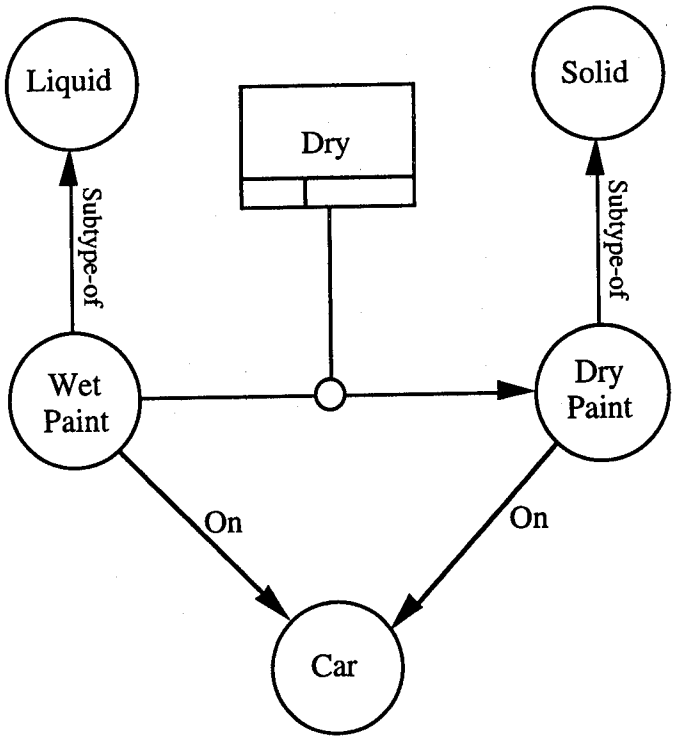


Figure 4-64. IDEF5 Schematic Involving OS Constructs

In this example, in addition to indicating the transition of a wet paint object to a dry paint object via a drying process, relation symbols are used to indicate that the states **Wet Paint** and **Dry Paint** are also related to other kinds, as indicated by the labels on the arrows. Again, despite the fact that relation symbols are similar to the arrows used in transition schematics and point-of-transition schematics, there is no danger of ambiguity, as the arrows in transition schematics always include an open circle.

4.2 The IDEF5 Elaboration Language

4.2.1 Overview

The IDEF5 Elaboration Language is a structured textual language that facilitates the direct capture of ontologies. The power of the language derives from its theoretically sound foundation and its expressively rich structure. The design of the IDEF5 Elaboration Language was motivated by the need to capture and represent complex ontology knowledge from a wide range of application

domains. While a graphical medium can help to visualize relatively simple ontology knowledge, its expressive limitation does not allow complex information to be easily expressed (if at all). Hence, the necessity for a more expressive medium was evident.

The perceived users of the IDEF5 Elaboration Language are knowledge engineers and systems analysts in collaboration with domain experts. The language is intended to be used along with the IDEF5 Schematic Language. The two languages will complement each other in a variety of different usage scenarios.

- The system analyst and domain expert record an initial ontology with the IDEF5 Schematic Language. This initial knowledge is analyzed, then recast into the (more structured) Elaboration Language format.[24]

- The ontology capture is done concurrently with the Elaboration Language and the Schematic Language, with one language complementing the other throughout the knowledge capture phase.

- The knowledge capture is done entirely with the Elaboration Language. The Schematic Language is used mainly for communication and post-acquisition visual analysis purposes.

As stated previously, the Elaboration Language can be used to capture the entire ontology. Therefore, some of its constructs duplicate the functions of the Schematic Language and the specification forms associated with the kinds, relations, properties, source material, source-statements, and ontology-terms. In general, the capture of ontology information using Elaboration Language constructs will be more difficult than using the Schematic Language and will require good knowledge and understanding of the language.

Subsection 4.2.2 describes the valid sentences of the languages and gives examples of use for each construct. For a detailed description of the language, refer to Appendix A, which contains the grammar for the language.

---

[24]If IDEF5 is used with an automated tool, some of the recast could be done automatically using the information represented in the schematic language.

4.2.2 Description of the Language

The syntax of the language uses a prefix notation and parentheses to delimit expressions. The alphabet for the language consists of the standard alphanumeric and punctuation characters. The *core* of the IDEF5 Elaboration Language, which enables the expression of axioms, is based on the Knowledge Interchange Format (KIF) [Genesereth 92]. In this subsection, the term *string* will refer to any finite sequence of characters and white spaces enclosed in double quotes. In the following subsections, the words that are reserved words in the language (i.e., that have a specific meaning in the language) will in appear in Courier font.

4.2.2.1 Constants, Variables, and Operators

The notion of a *word* is taken as a primitive of the IDEF5 Elaboration Language. A sentence in the language is composed of operators, constants, and, possibly, variables.

4.2.2.1.1 Constants

A constant is a word that is neither an operator nor a variable. Constants can be viewed as words denoting objects in the world. In the IDEF5 Elaboration Language, constants are divided into the following categories:

- **Ontology Constants** An ontology constant is a word denoting an ontology.
- **Logical Constants** A logical constant is a word denoting a truth value.
- **Individual Constants** An individual constant is a word denoting an individual.
- **Kind Constants** A kind constant is a word denoting a kind.
- **First-Order-Relation Constants** A first-order relation constant is a word denoting a first order predicate (i.e., a relation that holds between individuals).
- **Second-Order Relation Constants** A second-order relation constant is a word denoting a second-order predicate (i.e., a relation that holds between first order relations).
- **Relation Constants** A relation constant is either a first-order predicate constant or a second-order predicate constant.
- **Function Constants** A function constant is a word denoting a function.

- **Attribute Constants** An attribute constant is a word denoting an attribute.
- **Property Constants** A property constant is a word denoting a property.[25]
- **Object-State Constant** An object-state constant is a construct of the form kind:property, where kind is a kind constant and property is a property constant.
- **Source-Statement Constants** A source-statement constant is a word denoting a source-statement.
- **Ontology-Term Constants** An ontology-term constant is a word denoting a term in the ontology.
- **Source Constants** A source constant is a word denoting a source.
- **Note Constants** A note constant is a word denoting a note.

Examples of constants are **Mary** (an individual constant), **Car** (a kind constant), and **Transitive** (a property constant). In the description of terms, definitions, and sentences (see Subsections 4.2.2.2-4.2.2.4), the term *IDEF5-constant* will be used to refer to any constant that is neither an ontology-constant nor a logical constant. Constant may be prefixed by the name of an ontology when necessary to disambiguate a term (e.g., **manufacturing::technology** where **manufacturing** is the name of an ontology and **technology** is the name of a kind that is part of the manufacturing ontology).

4.2.2.1.2 Variables

A variable can be viewed as a one-place holder for a constant. In the IDEF5 Elaboration Language, a variable is a word whose first character is either ? or #. A word whose first character is ? is an *individual variable* (i.e., a one-place holder for an individual constant). A word whose first character is # is an *predicate variable* (i.e., a one-place holder for a predicate constant). Individual variables are used in quantifying over individual constants while predicate variables are used in quantifying over predicate constants. Examples of variables are **?x**, **?ind**, **#p**, and **#pred**.

---

[25]Properties are treated separately from relations in the IDEF5 elaboration language because of their roles as primary concepts of the method. A property can be a first-order predicate if it applies to individuals, or a second-order predicate if it applies to relations between individuals.

4.2.2.1.3 Operators

Operators are reserved words and characters that can be used to form complex expressions. The operators that are part of the IDEF5 Elaboration Language are presented in this Subsection. For a better understanding of the use of these operators, refer to Subsections 4.2.2.2 and 4.2.2.3.

- **Definition Operators** These operators are used in forming definitions. There are five definition operators in the Language: the `define-relation` operator is used in forming definitions of second and third-order predicates; the `define-function` operator is used in forming definitions of functions; the `define-individual` operator is used in forming definitions of individuals; finally, the operators `:=` and `:arg-types` are operators that are used in definitions (for more on these operators, see Subsections 4.2.2.2 and 4.2.2.3).

- **Term Operators** These operators are used to form complex terms. There are six such operators: `listof`, `setof`, `if`, `cond`, `the`, and `setofall`.

- **Sentence Operators** These operators are used to form complex sentences. The IDEF5 Elaboration Language includes common logical operators (`=`, `/=`, `not`, `and`, `or`, `implies`, `equiv`, `forall`, and `exists`), two modal operators (`nec` for necessary and `pos` for possibly), and a number of IDEF5-Specific operators. For a complete list of IDEF5-Specific operators, refer to Appendix A.

4.2.2.2 Terms

The IDEF5 Elaboration Language supports three types of expressions: terms, definitions, and sentences. Terms are used to denote objects. Examples of some terms are shown in Figure 4-65.

Terms are divided into the following categories.

- **IDEF5-Constants and Ontology Constants** All constants except logical constants are terms because they denote objects in the domain of discourse.

- **Variables** A variable is a term because it is a one-place holder for a constant.

- **Attribute Term** An attribute term is an expression of the form **(attribute-constant individual-constant)**. It denotes the object corresponding to the value of the attribute denoted by *attribute-constant* when applied to *individual-constant*. An example of an attribute term is the expression **(age-of Mary)** as in

Figure 4-65, in which **age-of** is an attribute constant and **Mary** is an individual constant. If Mary is 25 years of age, then the term denoted by the form **(attribute-constant individual-constant)** is the number **25**.

- **Function Term** A function term is an expression that contains a function constant followed by one or more terms. It denotes the object corresponding to the value of the function denoted by *function-constant* when applied to the argument *term*. For example, the functional term **(square 2)** denotes the object **4**.

- **List Term** A list term consists of the `listof` operator and one or more terms. The object denoted by a list term is the list of objects denoted by the given terms. The list term example presented in Figure 4-65 denotes the list containing the objects **blue**, **red**, and **white**.

- **Set Term** A set term consists of the `setof` operator followed by one or more terms. The object denoted by a set term is the set of objects denoted by the corresponding terms. For example, the set term **(setof employee manager)** denotes the set containing the objects **employee** and **manager**.

- **Logical Term** A logical term can denote one of several objects based on some given condition. It consists of either the `if` operator followed by a sentence and one or two terms, or the `cond` operator and a finite list of sentence-term pairs. In both cases, the sentence preceding a term denoting an object **O** represents the condition that must be met for the logical term to denote **O**. Two examples of logical terms are given in Figure 4-65. The terms might be used to denote the child sizes for a particular line of clothing. The first term denotes the constant **medium** if the attribute **weight** applied to a child is strictly less than 120 pounds, and the constant **large** otherwise. In this form of a logical term, the second term can be omitted. In such case, no default term is specified if the condition is not met. The second term allows for a finer grain characterization. It denotes the constant **small** if the attribute **weight** applied to some child is strictly less than 80 pounds, **medium** if the attribute value is strictly less than 120 pounds, and **large** otherwise.

- **Quantified Term** A quantified term is used to denote an object (or a set of objects) by specifying a condition that allows the identification of the object(s). It involves either the `the` operator or the `setofall` operator. A quantified term using the `the` operator denotes an object that satisfies the given condition, such as the first example shown in Figure 4-65, which denotes the object that Paul married. A

quantified term using the `setofall` operator denotes the set of objects that satisfy the given condition. An example of a quantified term using the `setofall` operator is the second example shown in Figure 4-65, which denotes the set of all objects that are married and are thirty years of age.

Attribute Term	(age-of Mary)
Functional Term	(square 2)
List Term	(listof blue red white)
Set Term	(setof employee manager)
Logical Term	1. (if (< (weight ?x)120) medium large) 2. (cond ((< (weight ?x) 80) small) ((< (weight ?x) 20) medium) ((>= (weight ?x) 120) large))
Quantified Term	1. (the (?x) (married Paul ?x)) 2. (setofall (?x) (and (married ?x) (= (age ?x) 30))

Figure 4-65. Examples of Terms in the IDEF5 Elaboration Language

4.2.2.3 Definitions

A definition is a type of expression used to define an individual, relation, or function constant. A definition can be complete or partial. A complete definition is used when a constant can be defined completely, while a partial definition is used when only limited information is available regarding a constant. For example, the expression **(define-individual origin := (listof 0 0))** defines the constant **origin** to be the list **(0,0)**, while the expression **(define-function F (?x ?y))** specifies only that the function F takes two arguments. A constant may have only one complete definition but several partial definitions.

There are three types of definitions illustrated in Figure 4-66. An individual definition is used to define an individual. In a complete individual definition, the name of the individual and a term defining the individual must be specified. In a partial definition, only the name of the individual must be specified. Optionally, a sentence can be included in a partial definition to restrict the definition of the individual.

A function definition is used to define a function. In a complete function definition, the name of the function, a list of variables (the number of variables in the list specifies the arity of the function), and a term that defines the function must be given. The types of the arguments expected by the function and the type of argument the function returns can also be specified. They are specified as a list whose elements are lists of kinds or object-states. Each list represents a legal set of argument types for the function. The last element in a list specifies the type of argument returned by the function. In a partial function definition, only the name of the function and a list of variables must be specified. Optionally, the types of the arguments expected by the function and a sentence restricting the definition of the function can be added.

Constant /Definitions	Complete	Partial
Individual	(define-individual origin := (listof 0 0))	(define-individual sun (rotates-around earth sun))
Function	(define-function integral-part (?x) :argument-type ((real integer)) := (the (?y) (and (integer ?y) (=< ?y ?x) (> ?y (- ?x 1))))	(define-function duration (?x) :argument-type ((process time-interval)) (number (duration ?x)))
Relation	(define-relation below (?x ?y) := (above ?y ?x))	(define-relation daughter-of (?x ?y) :argument-type ((female parent)) (=> (daughter-of ?x ?y) (child-of ?x ?y)))

Figure 4-66. Definitions in the IDEF5 Elaboration Language

A relation definition is used to define a relation. A complete relation definition contains the name of the relation, a list of variables (the number of variables in the list specifies the arity of the relation), and a sentence defining the relation completely. Optionally, the kinds of instances that can stand in the relation are specified (in the form of a list whose elements are lists of kinds or object-states). In a partial relation definition, only the name of the relation and a list of variables must be specified. Optionally, the kinds of instances that can stand in the relation (in the form of a list of kinds or object-states) and a sentence restricting the definition of the relation can be added.

4.2.2.4 Sentences

A sentence in the IDEF5 Elaboration Language expresses some fact about the constants in the ontology. There are seven types of sentences in the language, some of which are illustrated in Figure 4-67.

- **Logical Constants** A logical constant is a word denoting a truth value.
- **Equation** An equation is an expression consisting of the operator = and two terms.
- **Inequality** An inequality is an expression consisting of the symbol `/=` and two terms.
- **Relation Sentence** A relation sentence is an expression consisting of a relation constant or a function constant followed by one or more terms.
- **Logical Sentence** A logical sentence consists of a logical operator followed by the appropriate number of sentences. A logical sentence with the `not`, `pos`, or `nec` operator contains only one sentence for argument, while any other logical sentence has two sentences for argument.
- **Quantified Sentence** There are two types of quantified sentences: universally quantified sentences and existentially quantified sentences. A universally quantified sentence consists of the operator `forall` followed by one or more variables enclosed in parentheses and a sentence. Such a sentence is used to make a general statement about some class of objects. The universally quantified sentence in Figure 4-67, for example, states that all humans that are at least twenty-one years of age are adults. An existentially quantified sentence consists of the operator `exists` followed by one or more variables enclosed in parentheses and a sentence. An existentially quantified sentence allows the declaration of the existence of an object that possesses certain properties.
- **IDEF5-Specific Sentence** An IDEF5-specific sentence is a sentence that enables the introduction of constants according to the concept structure of the IDEF5 method. This type of sentence is described in detail in Subsection 4.2.2.5.

Equation	(= (age-of Paul) 30)
Inequality	(/= (gas-mileage truck) (gas-mileage car))
Relational sentences	(married-to Suzy James)
Logical sentence	(and (= (age Paul) 30) (not (married-to Paul Suzy)))
Quantified sentences	(forall (?x) (=> (and (human ?x) (> (age ?x) 21)) (adult ?x)))

Figure 4-67. Examples of Sentences in the IDEF5 Elaboration Language

4.2.2.5 IDEF5-Specific Sentences

There are eleven categories of IDEF5-specific sentences, each category corresponding to a concept in the IDEF5 method.

4.2.2.5.1 Ontology Constructs

There are eight ontology constructs, each of which provides a construct to express some information about an ontology. The nine constructs are explained in this Subsection and examples of some of the constructs are shown in Figure 4-68.

- **Ontology Sentence** This type of sentence is used to declare an ontology. An ontology declaration consists of the operator `I5-ontology` followed by an ontology constant or an individual variable.

- **Context Sentence** This type of sentence is used to declare the context in which an ontology is captured. A context declaration consists of the operator `I5-ontology-context` followed by an ontology constant or an individual variable and a string.

- **Viewpoint Sentence** This type of sentence is used to declare the viewpoint adopted to describe an ontology. A viewpoint declaration consists of the operator `I5-ontology-viewpoint` followed by an ontology constant or an individual variable and a string.

- **Purpose Sentence** This type of sentence is used to declare the purpose in capturing an ontology. A purpose declaration consists of the operator `I5-ontology-purpose` followed by an ontology constant or an individual variable and a string.

- **Project Sentence** This type of sentence is used to declare the project of which the ontology capture effort is a part. A project declaration consists of the operator `I5-ontology-project` followed by an ontology constant or an individual variable and a string.

- **Analyst Sentence** This type of sentence is used to declare the analyst responsible for the capture of the ontology. An analyst declaration consists of the operator `I5-ontology-analyst` followed by an ontology constant or an individual variable and a string.

- **Reviewer Sentence** This type of sentence is used to declare the person responsible for reviewing the ontology. A reviewer declaration consists of the operator `I5-ontology-reviewer` followed by an ontology constant or an individual variable and a string.

- **In Ontology Sentence** This type of sentence is used to declare that a constant is part of an ontology. It consists of the operator `I5-in-ontology` followed by an IDEF5 constant or variable and an ontology constant or an individual variable.

(I5-ontology shop_floor_control_system)

(I5-ontology-context shop_floor_control_system "This ontology is focused on understanding what knowledge is used to manage jobs and manufacturing resources in a job shop environment. The ontology will also describe key resource constraints that the shop floor control system must take into account.")

(I5-ontology-purpose shop_floor_control_system "The purpose of this ontology is to establish what information is actually required in a shop floor control system in a job shop environment. It will enable future analysts to determine what information and knowledge is critical to the success of controlling resources and jobs on a shop floor.")

(I5-ontology-viewpoint shop_floor_control_system "This ontology is described from the viewpoint of the cost center schedulers working in different cost centers throughout a company's precision gear manufacturing facility.")

Figure 4-68. Example of Ontology Constructs in the IDEF5 Elaboration Language

4.2.2.5.2 Kind Constructs

There are nine kind constructs, each of which provides a construct to express some information about a kind. The seven constructs are described in this Subsection and examples of some of the constructs are shown in Figure 4-69.

- **Kind Sentence** This type of sentence is used to declare a kind and consists of the operator `I5-kind` followed by a kind constant or a predicate variable. The example in Figure 4-69 shows a kind sentence which declares that **drilling** is a kind in a particular ontology.

- **Kind-Property Sentence** This type of sentence is used to associate a property with a kind. A property sentence consists of the operator `I5-kind-property` followed by a kind constant or predicate variable, a property constant or predicate variable, and, optionally, the reserved words `defining` and **essential**. The example shown in Figure 4-69 shows that the property **uses_drill_bit** (presumably declared beforehand) is a defining property of the kind **drilling**.

- **Kind-Attribute Sentence** This type of sentence is used to associate an attribute with a kind. An attribute sentence consists of the operator `I5-kind-attribute` followed by a kind constant or predicate variable and an attribute constant or predicate variable. The kind-attribute sentence shown in Figure 4-69 declares that **drill_bit_type** is an attribute of the kind **drilling**. It is assumed that **drill_bit_type** was previously declared to be an attribute.

- **Kind-Description Sentence** This type of sentence is used to associate a description string with a kind. A kind description sentence consists of the operator `I5-kind-description` followed by a kind constant or predicate variable and a string.

- **Kind-Synonyms Sentence** This type of sentence is used to declare the ontology-terms that are synonyms to a kind constant. A kind synonyms declaration consists of the operator `I5-kind-synonyms` followed by a kind constant or predicate variable and one or more ontology-term constants or individual variables enclosed in parentheses.

- **Referenced-Relations Sentence** This type of sentence is used to declare the relations in which a kind is involved. A referenced-relations declaration consists of the operator `I5-referenced-relations` followed by a kind-constant or predicate variable and one or more relation constants or predicate variables enclosed in parentheses.

- **Subkind Sentence** This type of sentence is used to declare a kind as a subkind of another. A subkind sentence consists of the operator `I5-subkind-of` followed by a kind-constant or predicate variable and a kind-constant or predicate variable.

- **Object-State Sentence** This type of sentence is used to declare an object state and consists of the operator `I5-object-state` followed by an object-state constant or a predicate variable.

- **Process Sentence** This type of sentence is used to declare a process and consists of the operator `I5-process` followed by a process constant or a predicate variable.

```
(I5-kind drilling)

(I5-kind-description drilling "Drilling is a type of cutting
   operation.")

(I5-kind-property drilling uses_drill_bit defining)

(I5-kind-attribute drilling drill_bit_type )
```

Figure 4-69. Examples of Kind Constructs in the IDEF5 Elaboration Language

4.2.2.5.3 Property Constructs

Property constructs are used to provide information about properties. There are three types of property constructs. Some examples of property constructs are given in Figure 4-70.

- **Property Sentence** A property sentence is used to declare a property. It consists of the operator `I5-property` followed by a property constant or a predicate variable. The property sentence shown in Figure 4-70 declares the property **made_in_Detroit**.

- **Property-Description Sentence** This sentence can be used to describe a property. It consists of the operator `I5-property-description` followed by a property constant or a predicate variable and a string.

- **Has-Property Sentence** This sentence is used to declare that an individual possesses a given property. It consists of the operator `I5-has-property` followed by an individual constant or variable and a property constant or predicate variable. The has-property sentence shown in Figure 4-70 declares the property **made_in_Detroit** to be a property of the individual **CB-J27-S121.**

4.2.2.5.4 Individual Constructs

Individual constructs are used to declare information about individuals. There are three individual constructs in the IDEF5 Elaboration Language. Examples of individual constructs are given in Figure 4-70.

- **Individual Sentence** this sentence is used to declare individuals. It consists of the operator `I5-individual` followed by an individual constant or variable. The

individual sentence shown in Figure 4-70 declares **CB-J27-S121** to be an individual.

- **Individual-Description Sentence** This sentence is used to associate a description string with an individual. It consists of the operator `I5-individual-description` followed by an individual constant or variable and a string.

- **Is-of-kind Sentence** This sentence is used to declare that an individual is of a given kind. It consists of the operator `I5-is-of-kind` followed by an individual constant or variable and a kind constant or a predicate variable. The is-of-kind displayed in Figure 4-70 declares the individual **CB-J27-S121** to be of kind **car-body**.

```
(I5-individual CB-J27-S121)
(I5-property made_in_Detroit)
(I5-has-property CB-J27-S121 made_in_Detroit)
(I5-is-of-kind CB-J27-S121 car-body)
```

Figure 4-70. Examples of Individual and Property Constructs

4.2.2.5.5 Attribute Constructs

Attribute constructs are used to provide information about attributes. There are three types of attribute constructs.

- **Attribute Sentence** This sentence is used to declare attributes. It consists of the operator `I5-attribute` followed by an attribute constant or a predicate variable and an attribute type. An attribute type may be one of the following: a list term, a set term, or a kind constant. An example of an attribute sentence is the sentence **(I5-attribute dimensions (listof real))** which declares **dimensions** to be an attribute of type a list of real numbers.

- **Attribute-Description Sentence** This sentence is used to describe an attribute. It consists of the operator `I5-attribute-description` followed by an attribute constant or a predicate variable and a string.

- **Attribute-Applies-to Sentence** This sentence is used to declare that an attribute applies to a given individual. It consists of the operator `I5-attribute-applies-to` followed by an attribute constant or a predicate variable and an individual constant or variable. An example of an attribute-applies-to sentence is the sentence **(I5-attribute-applies-to dimensions CB-J27-S121)** which declares **dimensions** to be an attribute of the individual **CB-J27-S121**.

4.2.2.5.6 Relation Constructs

Relation constructs are used to provide information about relations. There are four relation constructs, some of which are illustrated in Figure 4-71.

- **Relation Sentence** This sentence is used to declare relations. It consists of the operator `I5-relation` followed by a relation constant or predicate variable. The relation sentence shown in Figure 4-71 declares **part-of** to be a relation.

- **Relation Arity Sentence** This sentence is used to declare the arity of a relation. It consists of the operator `I5-relation-arity` followed by a relation constant or predicate variable and a positive integer or individual variable. The relation sentence shown in Figure 4-71 declares **part-of** to be a relation of arity 2.

- **Relation-argument-type Sentence** This sentence is used to declare the kind of individuals that can stand in a relation. It consists of the operator `I5-rel-arg-type` followed by a relation constant or predicate variable and a list whose elements are lists of kinds and/or predicate variables. Each list of kinds represents a set of legal argument kinds for the relation. For example, the relation-argument-type sentence given in Figure 4-71 declares that instances of the kind spark-plug can stand in the part-of relation with instances of the kind engine, and that instances of the kind engine can stand in the part-of relation with instances of the kind car.

- **Relation Description Sentence** This sentence is used to describe a relation. It consists of the operator `I5-relation-description` followed by a relation constant or a predicate variable and a string.

```
(I5-relation part-of)
(I5-relation-arity part-of 2)
(I5-rel-arg-type part-of ((spark-plug engine) (engine car)))
```

Figure 4-71. Examples of Relation Constructs

4.2.2.5.7 Function Constructs

Function constructs are used to provide information about functions. There are four function constructs, some of which are illustrated in Figure 4-72.

- **Function Sentence** This sentence is used to declare functions. It consists of the operator `I5-function` followed by a function constant or predicate variable. The example in Figure 4-72 declares **warranty-exp** to be a function.

- **Function-Arity Sentence** This sentence is used to declare the arity of a function (i.e., the number of arguments that the function can take). It consists of the operator `I5-function-arity` followed by a function constant or predicate variable and a positive integer or individual variable. The example in Figure 4-72 declares that the function **warranty-exp** takes one argument (i.e., is of arity 1).

- **Function-Argument-Type Sentence** This sentence is used to declare the argument types for a function. It consists of the operator `I5-fct-arg-type` followed by a function constant or predicate variable and a list of kinds and/or predicate variables. The example in Figure 4-72 declares that the function **warranty-exp** takes an instance of date for argument and returns an instance of date.

- **Function-Description** This sentence is used to describe a function. It consists of the operator `I5-function-description` followed by a function constant or a predicate variable and a string.

(I5-function warranty-exp)

(I5-function-arity warranty-exp 1)

(I5-function-description warranty-exp "This function returns the expiration date of the warranty on a car battery given the date shown on the battery.")

(I5-fct-arg-type warranty-exp ((date date))

Figure 4-72. Examples of Function Constructs

4.2.2.5.8 Source Constructs

Source constructs are used to provide information about source materials. There are nine source constructs, some of which are illustrated in Figure 4-73.

- **Source Sentence** A source sentence is used to declare a source material. It consists of the operator `I5-source` followed by a source constant or an individual variable. An example of a source sentence is given in Figure 4-73. The sentence declares **Chang_TC** as a source material.

- **Source-Description Sentence** A source description sentence is used to describe a source material. It consists of the operator `I5-source-description` followed by a source constant or an individual variable and a string.

- **Collected-from Sentence** This sentence is used to declare who the source material was collected from. It consists of the operator `I5-collected-from` followed by a source constant or an individual variable and a string.

- **Collected-by Sentence** This sentence is used to declare by whom the source material was collected. It consists of the operator `I5-collected-by` followed by a source constant or an individual variable and a string.

- **Source-Abstract Sentence** This sentence is used to give an abstract of the source material. It consists of the operator `I5-source-abstract` followed by a source constant or an individual variable and a string. An example of a source-abstract sentence is given in Figure 4-73. The sentence specifies the source-abstract for the source material **Chang_TC**.

- **Source-Purpose Sentence** This sentence is used to declare the purpose of a source material. It consists of the operator `I5-source-purpose` followed by a source constant or an individual variable and a string. An example of a source-purpose sentence is given in Figure 4-73. The sentence specifies the purpose of using the source material **Chang_TC**.

- **Support-Ontology-Terms Sentence** This sentence is used to declare the ontology-terms that are supported by a source material. It consists of the operator `I5-support-ontology-term` followed by a source constant or an individual variable and one or more ontology-terms or individual variables enclosed in parentheses.

- **Support-Statement Sentence** This sentence is used to declare the source-statements in the ontology that are supported by a source material. It consists of the operator `I5-support-statement` followed by a source constant or an individual variable and one or more source-statements or individual variables enclosed in parentheses. An example of a support-statement sentence is given in Figure 4-73. The sentence declares the statements **S1**, **S2**, and **S3** to be supported by the source material **Chang_TC**.

- **Has-Supporting-Sources Sentence** This sentence is used to declare that an IDEF5 constant is supported by one or more source materials. It consists of the operator `I5-has-supporting-sources` followed by an IDEF5 constant or a variable and one or more source constants or an individual variable enclosed in parentheses.

```
(I5-source Chang_TC)

(I5-source-purpose Chang_TC "Description of general terms in
    manufacturing domains.")

(I5-source-abstract Chang_TC "Role of computers in the
    planning, control and scheduling of manufacturing
    processes; computer controlled manufacturing systems with
    emphasis on the design and integration of hardware and
    software systems.")

(I5-support-statement Chang_TC (S1 S2 S3)
```

Figure 4-73. Examples of Source Constructs

4.2.2.5.9 Source-Statement Constructs

Source-statement constructs are used to provide information about source-statements. There are four such constructs, some of which are illustrated in Figure 4-74.

- **Source-Statement Sentence** This sentence is used to declare a source-statement. It consists of the operator `I5-source-statement` followed by a source-statement constant or an individual variable. An example of a source-statement sentence is given in Figure 4-74. The sentence declares **design_questions** as a source-statement.

- **Source-Statement-Description** This sentence provides the text associated with a source-statement and/or provides a description of the source-statement. It consists of the operator `I5-source-statement-description` followed by a source-statement constant or an individual variable and a string. An example of a source-statement-description sentence is given in Figure 4-74. The sentence provides the text associated with the **design_questions** source-statement.

- **Status Sentence** This sentence is used to declare the status of a source-statement. It consists of the operator `I5-status` followed by a source-statement constant or an individual variable and one of the following operators: active-original, active-derived, retired-original, or retired-derived. An example of a status sentence is given in Figure 4-74. The sentence declares the status of the **design_questions** source-statement as active and original.

- **Has-Original-Statement Sentence** This sentence is used to declare the original source-statement of a derived source-statement. It consists of the operator `I5-has-original-statement` followed by a source-statement constant or an individual variable and a source-statement constant or individual variable.

```
(I5-source-statement design_questions)

(I5-source-statement-description design_questions "Design
    questions should consider the most appropriate material
    removal process which depends on volume, batch size,
    accuracy, finish, materials, and cost constraints.")

(I5-status design_questions active/orginal)
```

Figure 4-74. Examples of Source-statement Constructs

4.2.2.5.10 Ontology-Term Constructs

Ontology-term constructs are used to provide information about ontology-terms. There are three such constructs, some of which are illustrated in Figure 4-75.

- **Ontology-Term Sentence** This sentence is used to declare an ontology-term. It consists of the operator `I5-ontology-term` followed by an ontology-term constant or an individual variable. An example of a ontology-term sentence is given in Figure 4-75. The sentence declares **Perishable_Tooling** as an ontology-term.

- **Ontology-Term description Sentence** This sentence is used to describe an ontology-term and consists of the operator `I5-ontology-term-description` followed by an ontology-term constant or an individual variable and a string. An example of an ontology-term sentence is given in Figure 4-75. The sentence describes the ontology-term **Perishable_Tooling**.

- **Used-in-Statements Sentence** This sentence is used to declare the statements used by an ontology-term. It consists of the operator `I5-used-in-statements` followed by an ontology-term constant or an individual variable and one or more source-statement constant or individual variables enclosed in parentheses. An example of an ontology-term sentence is given in Figure 4-75. The sentence declares that the ontology-term **Perishable_Tooling** is used in the source-statement **SS39**.

```
(I5-ontology-term Perishable_Tooling)

(I5-ontology-term-description Perishable_Tooling "Tools
    with a limited usable life due to wear.")

(I5-used-in-statements Perishable_Tooling (SS39))
```

Figure 4-75. Examples of Ontology-Term Constructs

4.2.2.5.11 Note Constructs

Note constructs are used to provide information about notes. There are three such constructs and they are described as follows.

- **Note Sentence** This sentence is used to declare a note. It consists of the operator `I5-note` followed by a note constant or an individual variable.

- **Note-Description Sentence** This sentence is used to describe a note. It consists of the operator `I5-note-description` followed by a note constant or an individual variable and a string.

- **Has-Note Sentence** This sentence is used to "attach" a note to an IDEF5 constant. It consists of the operator `I5-has-note` followed by an IDEF5 constant or a variable and a note constant or an individual variable.

4.2.2.6 Predefined Relations

The relation constants listed in this Subsection are predefined in the language and, hence, can be used in sentences.

- **part-of** This relation constant denotes the "part of" relation. It is partially defined as a first-order relation of arity 2 and takes two instances of kinds as arguments.

- **transitions-to** This relation constant denotes the "transitions to" relation that can be found in a Basic State Transition Schematic where it is represented as an object-state transition link without a process symbol attached to it. It is partially defined as a first-order relation of arity 2 and takes two instances of kinds as arguments.

- **transitions-during** This relation constant denotes the "transitions during" relation that can be found in the Basic State Transition Schematics where it is represented as a state transition link without a process symbol attached to it. It is partially defined as a first-order relation of arity 3 and takes two instances of kinds and an instance of a process as arguments.

- **inst-transitions-to** This relation constant denotes the "transitions instantaneously to" relation that can be found in the Basic State Transition Schematics where it is represented as an instantaneous state transition link without a process symbol attached to it. It is partially defined as a first-order relation of arity 2.

- **inst-transitions-during** This relation constant denotes the "transitions instantaneously during" relation that can be found in the Basic State Transition Schematics where it is represented as an instantaneous state transition link with a process symbol attached to it. It is partially defined as a first-order relation of arity 3 and takes two instances of kinds and an instance of a process as arguments.

- **s-transitions-to** This relation constant denotes the "strongly transitions to" relation that can be found in the Strong Transition Schematics where it is represented as a strong transition link without a process symbol attached to it. It is partially defined as a first-order relation of arity 2 and takes two instances of kinds as arguments.

- **s-transitions-during** This relation constant denotes the "strongly transitions during" relation that can be found in the Basic State Transition Schematics where it is represented as a strong state transition link with a process symbol attached to it. It is partially defined as a first-order relation of arity 3 and takes two instances of kinds and an instance of a process as arguments.

- **inst-s-transitions-to** This relation constant denotes the instantaneous **s-transitions-to** relation. It is partially defined as a first-order relation of arity 2 and takes two instances of kinds as arguments.

- **inst-s-transitions-during** This relation constant denotes the instantaneous **s-transitions-to** relation. It is partially defined as a first-order relation of arity 3 and takes two instances of kinds and an instance of a process as arguments.

- **in-state-throughout** This relation constant is used to specify that an individual stays in a given state throughout a process. It is partially defined as a first-order relation of arity 2 and takes an instance of a kind and an instance of a process as arguments.

Appendices, Bibliography, Glossary

Appendix A: IDEF5 Relation Library

This appendix describes the IDEF5 relation library. The library is a knowledge-rich repository made of a set of definitions and characterizations of commonly used relations. It provides a repository of formally defined and characterized relations that can be reused and customized. The motivation for this library developed from an analogy with software engineering. Often in software development, the same kinds of routines are used again and again in different programs by (in general) different programmers. In earlier times, great amounts of time and effort were lost for lack of the ability to reuse work. Recognition of this problem over time has led to the development of extensive *libraries* that contain frequently used routines which programmers can simply call straight into their programs. Such libraries have eliminated the need to duplicate the functionality of existing code. The development of ontologies will face the same sort of problem (and solution). The same or similar relations will likely appear in a number of different ontologies. A library of relations such as the one presented here will enable modelers to reuse and customize relations that have been defined in previously captured ontologies. The library can also be used as a reference for the different ways to define and characterize relations and illustrative examples of the use of the IDEF5 elaboration language. All definitions and characterizing axioms in the library are written using the IDEF5 elaboration language. Finally, the library is extensible in that any relation that has been formally defined and characterized may be added to it.

The IDEF5 library relations are grouped into the following seven categories.

1. Classification Relations (including class inclusion relations).
2. Meronymic Relations.
3. Temporal Relations.
4. Spatial Relations.
5. Influence Relations.
6. Dependency Relations.
7. Case Relations.

Figure A-1 illustrates the IDEF5 relations categorization.

This appendix is organized in the following manner. For each type of relation shown in Figure A-1, an overview describing the relation type is given, the formal definitions of the relations that are members of that type are provided, and the relations are formally characterized. Most axioms that are part of the characterizations are followed by a brief explanation. The

formal definitions are numbered using the letter "D" followed by a numeral, while axioms are numbered using the letters "Ax" followed by a numeral.

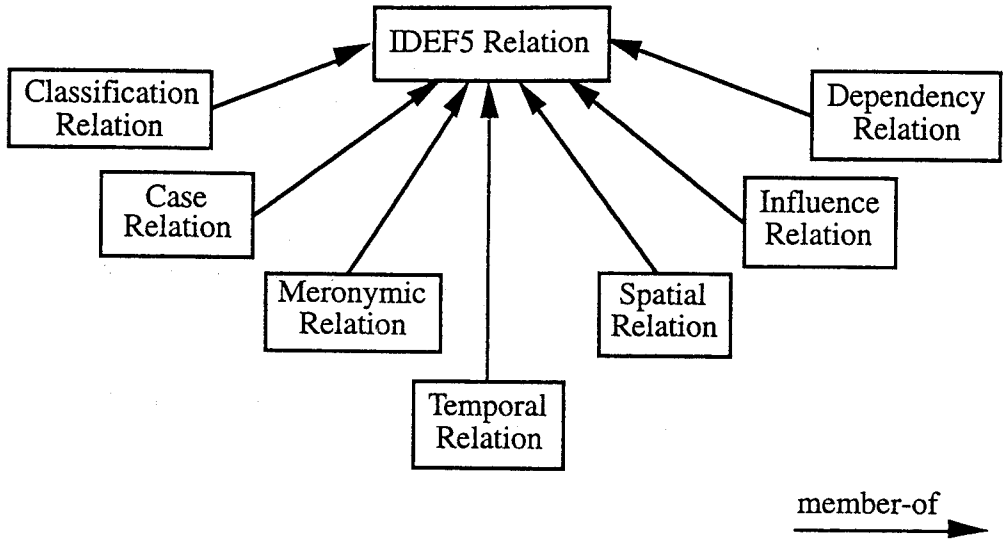


Figure A-1. Overview of the IDEF5 Library Relations

A.1 Classification Relations

A.1.1 Overview

The classification relations are targeted at capturing the intuitive semantics of the **is-a** relation and include the categorization relations kind and type and the class inclusion relations. A categorization relation allows the specification that an object belongs to a certain kind or type. At the most general level, a class inclusion relation relates two types (or kinds) when one type subsumes the other (relations subkind and subtype). At a more detailed level, the meaning of a class inclusion relation can be determined from knowledge of the basic nature of related objects and the context of use of the relation. Five specialized class inclusion relations, which were chosen for their prominence in AI research results, are provided [Winston, et. al. 87]. The distinction between the five more specific class inclusion relations is based on the basic nature of the related kinds or types as follows.

- **Functional-Inclusion** This relation is used to relate kinds whose relationship is functional in nature. **A chair is-a piece of furniture** and **A hammer is a tool** are examples of functional-inclusion relations.

- **State-Inclusion** This relation is used to relate kinds whose relationship involves a state or condition. **Polio is a disease** and **Hate is an emotion** are examples of state-inclusion relations.

- **Activity-Inclusion** This relation is used to relate kinds whose relationship involves an activity. **Tennis is a sport** and **Murder is a crime** are examples of activity-inclusion relations.

- **Action-Inclusion** This relation is used to relate kinds whose relationship involves an action. **Lecturing is a form of talking** and **Frying is a form of cooking** are examples of functional-inclusion relations.

- **Perceptual-Inclusion** This relation is used to relate kinds whose relationship is perceptual in nature. **A cat is a mammal** and **An apple is a fruit** are examples of perceptual-inclusion relations.

A.1.2 Relation Definition

As explained in Section 2, a kind contains instances that are similar in some, possibly arbitrary, way. It is required only that an object possess at least one defining property of the kind for it to be an instance of that kind. A type, on the other hand, regroups objects that share the same properties. An object is an instance of a type if and only if it has all the properties of the type. This section contains the formal definitions of the classification relations. The five more specific class inclusion relations can take either two kinds or two types as arguments.

D.1 (defrelation type (#t)

D.2 (defrelation subkind-of (#x #y)
:argument-type ((I5-kind I5-kind)))

D.3 (defrelation subtype-of (#x #y)
:argument-type ((type type)))

D.4 (defrelation functional-inclusion (?x ?y)
:argument-type ((I5-kind I5-kind) (type type)))

D.5 (defrelation state-inclusion (?x ?y)
:argument-type ((I5-kind I5-kind) (type type)))

D.6 (defrelation activity-inclusion (?x ?y)
:argument-type ((I5-kind I5-kind) (type type)))

D.7 (defrelation action-inclusion (?x ?y)
:argument-type ((I5-kind I5-kind) (type type)))

D.8 (defrelation perceptual-inclusion (?x ?y)
:argument-type ((I5-kind I5-kind) (type type)))

A.1.3 Relation Characterization

In this section, the relations defined in Subsection A.1.2 are characterized. Axioms Ax.1 to Ax.5 are general statements about the classifications relations. Axioms Ax.6 to Ax.8 state some of the properties of the classification relations. Properties that are common to all classification relations need only be stated for the kind and subkind relations because all other relations are specializations of these two relations.

A.1.3.1 General axioms

Ax.1 (forall (?x #y)
(=> (and (I5-kind #y) (instance-of ?x #y))
(exists (#p) (and (I5-kind-property #y #p defining)
(I5-has-property ?x #p)))))

Ax.2 (forall (?x #y)
(<=> (and (type ?y) (I5-is-of-kind ?x #y))
(forall (?p) (= > (I5-kind-property #y #p defining) (I5-has-property ?x #p))))

Ax.3 (forall (#x) (=> (type #x) (kind #x)))

- "Type" is a specialization of "kind," hence every type is also a kind.

Ax.4 (forall (#x #y #p)
(=> (and (subtype #x #y) (I5-kind-property #y #p defining))
(I5-kind-property #x #p defining)))

- If a type T1 is a subtype of a type T2 and P is a defining property for T2, then P is also a defining property for T1.

Ax.5 (forall (#x #y #p) (=> (and (subkind #x #y) (I5-kind-property #y #p essential))
(I5-kind-property #x #p essential)))

- If a kind K1 is a subkind of a kind K2 and P is an essential property of K2, then P is also an essential property of K1.

Ax.6 (forall (#x #y) (=> (subkind #x #y)
(forall (?z) (=> (is-of-kind ?z #x) (is-of-kind ?z #y)))))

- If a kind K1 is a subkind of a kind K2, then all instances of K1 are also instances of K2. Note that from this axiom and Axiom Ax.1, it can deduced that the same axiom holds for subtypes.

Ax.7 (forall (#x #y)
(=> (or (functional-inclusion #x #y) (activity-inclusion #x #y)
(state-inclusion #x #y) (action-inclusion #x #y)
(perceptual-inclusion #x #y) (subtype #x #y))
(subkind #x #y)))

- The subtype, functional-inclusion, activity-inclusion, state-inclusion, action-inclusion, and perceptual-inclusion relations are all specializations of the subkind relation.

A.1.3.2 Reflexivity

A relation has the property of reflexivity if every object stands in the relation with itself. For example, the relation **knows**, which takes humans as arguments, is reflexive because each human knows himself/herself. The following axiom states that the **subkind** relation is reflexive.

Ax.8 (forall (#x) (subkind #x #x))

A.1.3.3 Antisymmetry[26]

A relation R is antisymmetric if the fact that an object A is related to an object B through R and B is also related to A through R implies that A and B denote the same object. An example of an antisymmetric relation is the relation ≥ (greater than or equal to). If x is greater than or equal to

---

26A relation R is symmetric if for every objects A and B, such that A is related to B through R, B is also related to A through R.

y and y is greater than or equal to x, then x is equal to y. The following axiom states that the **subkind** relation is antisymmetric.

Ax.7 (forall (#x #y)
(=> (and (subkind #x #y) (subkind #y #x)) (= #x #y)))

A.1.3.4 Transitivity

A relation R is transitive if for all objects A, B, and C, such that A is related to B through R and B is related to C through R, A is related to C through R. An example of a transitive relation is the ≥ relation. If A≥B and B≥C, then A≥C. The following axiom states that the **subkind** relation is transitive.

Ax.9 (forall (#x #y #z) (=> (and (subkind #x #y) (subkind #y #z)) (subkind #x #z)))

A.2 Meronymic Relations

A.2.1 Overview

Meronymic relations are used to describe part-of relationships and are introduced in Section 4. Figure A-2 shows a partial taxonomy of meronymic relations. Physical and conceptual **part of** relationships are first distinguished. There are four physical **part of** relations. The **place-within** relation is used to relate geographical objects. It relates a geographical object and the area where the object is located. Examples of the use of this relation are **The Everglades are part of Florida** and **The Alps are part of Europe**. The **component-of** relation is used to relate an object and one of its components. The object is considered an integral whole that is divided into its components. Examples of the use of this relation are **A chapter is a part of a book** and **A wheel is a part of a bicycle**. The **stuff-of** relation is used to describe that an object is partly made of some material. Examples of the stuff-of relation are **A bike is partly steel** and **A chair is partly wood**. The **portion-of** relation describes the relationships between two similar objects, one being included in the other. Examples of **portion-of** relations are **A slice is part of a pie** and **A yard is part of a mile**.

There are two conceptual **part-of** relations. The **member-of** relation describes the fact that an object is a member of some collection. It is important to note that this relation differs from the classification relations described in Section 4. The **member-of** relation does not require any similarity between the members of a collection of objects except their membership to that

collection. Examples of the use of the **member-of** relation are **Cards are part of a deck** and **A tree is part of a forest**. The **activity-within** relation describes the features or phases of activities. Examples of the use of that relation are **Paying is part of shopping** and **Dating is part of adolescence**.

A.2.2 Relation Definition

In this subsection, the meronymic relations described in Subsection A.2.1 are formally defined. Most definitions consist simply of the declaration of the relation. The inverse relations of the relations shown in Figure A-2 are also defined.

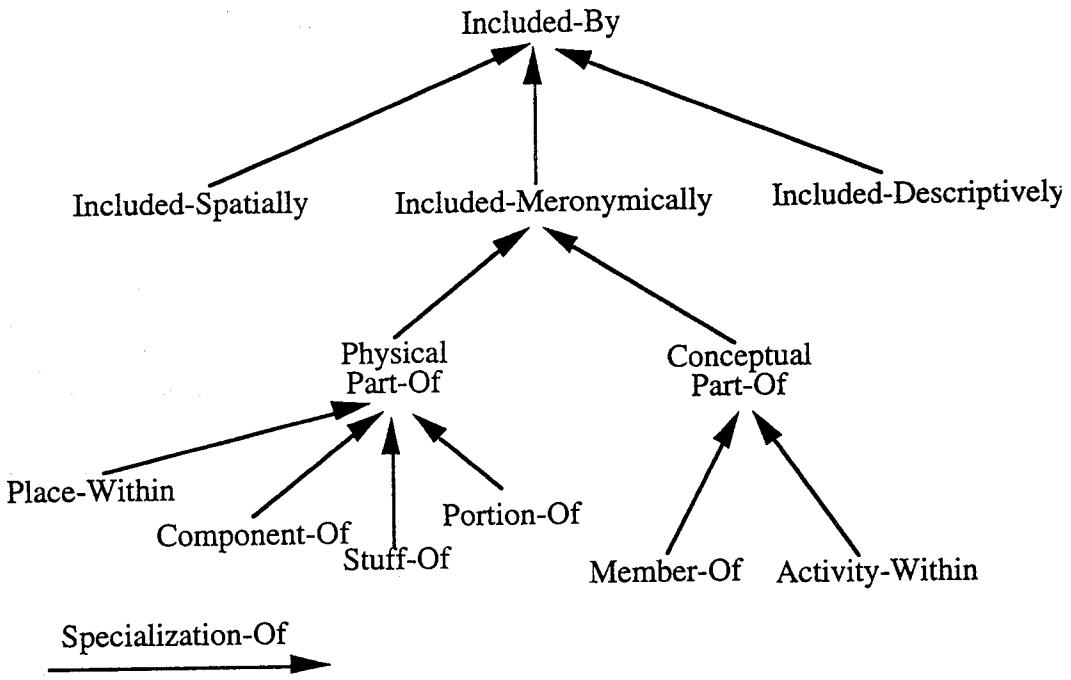


Figure A-2. A Partial Taxonomy of Meronymic Relations

D.9 (defrelation member-of (?x ?y))

D.10 (defrelation has-member (?x ?y) := (member-of ?y ?x))

- The relation has-member is defined as the inverse relation of the relation member-of.

D.11 (defrelation activity-within (?x ?y))

D.12 (defrelation contains-activity (?x ?y) := (activity-within ?y ?x))

- The relation contains-activity is defined as the inverse relation of the relation activity-within.

D.13 (defrelation made-of (?x ?y) :=> (material ?y))

D.14 (defrelation makes-up (?x ?y) := (made-of ?y ?x))

- The relation makes-up is defined as the inverse relation of the relation made-of.

D.15 (defrelation stuff-of (?x ?y) :=> (material ?y))

D.16 (defrelation makes-in-part (?x ?y) := (stuff-of ?y ?x))

- The relation makes-in-part is defined as the inverse relation of the relation stuff-of.

D.17 (defrelation portion-of (?x ?y)
(forall (?x ?y ?z) (exists (?z) (and (not (equal ?x ?z) (portion-of ?z ?y)))))

- If x is a portion of y, then there exists some other material z of which x is composed.

D.18 (defrelation has-portion (?x ?y) := (portion-of ?y ?x))

- The relation has-portion is defined as the inverse relation of the relation portion-of.

D.19 (defrelation component-of (?x ?y))

D.20 (defrelation has-component (?x ?y) := (component-of ?y ?x))

- The relation has-component is defined as the inverse relation of the relation component-of.

D.21 (defrelation place-within (?x ?y))

D.22 (defrelation contains-place (?x ?y) := (place-within ?y ?x))

- The relation contains-place is defined as the inverse relation of the relation place-within.

D.23 (defrelation physical-part-of (?x ?y)
:= (or (component-of ?x ?y) (stuff-of ?x ?y) (portion-of ?x ?y) (place-within ?x ?y)))

- The relations component-of, stuff-of, portion-of, and place-within are all physical-part-of relations and there exists no other physical-part-of relation.

D.24 (defrelation conceptual-part-of (?x ?y)
:= (or (member-of ?x ?y) (feature-of ?x ?y)))

- The relations member-of and feature-of are all conceptual-part-of relations and there exists no other conceptual-part-of relation.

D.25 (defrelation part-of (?x ?y)
:= (or (physical-part-of ?x ?y) (conceptual-part-of ?x ?y)))

- The part-of relations consist of the physical-part-of relations and the conceptual part-of relations.

A.2.3 Relation Characterization

In this subsection, the relations defined in Subsection A.2.2 are characterized. The characterization is organized according to common properties of relations.

A.2.3.1 Mutual Exclusivity

A set of n-ary relations (i.e., relations with n arguments) has the property of mutually exclusivity if, given n objects, either the objects stand in none of the relations or they stand in exactly one relation. The axioms below formally state that all meronymic relations are mutually exclusive.

Ax.10 (forall (?x ?y) (=> (component-of ?x ?y)
(and (not (conceptual-part-of ?x ?y)) (not (stuff-of ?x ?y))
(not (portion-of ?x ?y)) (not (place-within ?x ?y)))))

Ax.11 (forall (?x ?y) (=> (stuff-of ?x ?y)
(and (not (conceptual-part-of ?x ?y)) (not (component-of ?x ?y))
(not (portion-of ?x ?y)) (not (place-within ?x ?y)))))

Ax.12 (forall (?x ?y) (=> (portion-of ?x ?y)
(and (not (conceptual-part-of ?x ?y)) (not (component-of ?x ?y))
(not (stuff-of ?x ?y)) (not (place-within ?x ?y)))))

Ax.13 (forall (?x ?y) (=> (place-within ?x ?y)
(and (not (conceptual-part-of ?x ?y)) (not (portion-of ?x ?y)
(not (component-of ?x ?y)) (not (stuff-of ?x ?y)))))

Ax.14 (forall (?x ?y) (=> (member-of ?x ?y)
(and (not (physical-part-of ?x ?y)) (not (activity-within ?x ?y)))))

Ax.15 (forall (?x ?y) (=> (activity-within ?x ?y)
(and (not (physical-part-of ?x ?y)) (not (member-of ?x ?y)))))

A.2.3.2 Irreflexivity

An irreflexivity axiom for a relation states that no object can stand in the relation with itself. The following axiom formally states that all meronymic relations are irreflexive.

Ax.16 (forall (?x) (not (part-of ?x ?x))

A.2.3.3 Asymmetry

A relation **R** is asymmetric if the fact that an object **A** is related to an object **B** through **R** implies that **B** does not stand in that relation with **A**. The axiom below formally states that all meronymic relations are asymmetric.

Ax.17 (forall (?x ?y) (=> (part-of ?x ?y) (not (part-of ?y ?x))))

A.2.3.4 Transitivity

The axiom below formally states that all **physical-part-of** relations are transitive.

Ax.18 (forall (?x ?y ?z) (=> (and (physical-part-of ?x ?y) (physical-part-of ?y ?z))
(physical-part-of ?x ?z)))

A.2.3.5 Miscellaneous

The meronymic relations may be characterized further with three properties: separability, functionality, and homeomerosity [Winston, et. al. 87]. These properties enable further-grained differentiation of meronymic relations. The separability property applies to part-of relations in which objects can be physically separated, at least in principle, from the whole to which they are connected. Objects that participate in a functional relation have a functional role with respect to the whole. Objects participating in a homeomerous relation are similar to each other and to the whole to which they belong. These three properties are formally defined below. Figure A-3 summarizes the properties for each meronymic relation. In the table, the symbol "-" indicates that the corresponding relation does not have the property, while the symbol "+" indicates that the corresponding relation has the property. The formal IDEF5 elaboration language statements for the three properties of separability, functionality, and homeomerosity are provided.

Meronymic Relations	Separable	Functional	Homeomerous
place-within	-	-	-
component-of	+	+	-
stuff-of	-	-	-
portion-of	+	-	+
member-of	+	-	-
activity-within	-	+	-

Figure A-3. Special Properties of Meronymic Relations

Ax.19 (property separable)

Ax.20 (property homeomerous)

Ax.21 (property functional)

Ax.22 (has-property member-of separable)

Ax.23 (has-property portion-of separable)

Ax.24 (has-property component-of separable)

Ax.25 (not (has-property activity-within separable))

Ax.26 (not (has-property stuff-of separable))

Ax.27 (not (has-property place-within separable))

Ax.28 (not (has-property member-of functional))

Ax.29 (not (has-property portion-of functional))

Ax.30 (has-property component-of functional)

Ax.31 (has-property activity-within functional)

Ax.32 (has-property stuff-of functional)

Ax.33 (not (has-property place-within functional))

Ax.34 (not (has-property member-of homeomerous))

Ax.35 (has-property portion-of homeomerous)

Ax.36 (not (relation-property component-of homeomerous))

Ax.37 (not (relation-property activity-within homeomerous))

Ax.38 (relation-property stuff-of homeomerous)

Ax.39 (not (relation-property place-within homeomerous))

A.3 Temporal Relations

A.3.1 Overview

This subsection contains the definitions and characterizations of a set of commonly occurring temporal relations. These relations should be viewed as part of a time ontology that contains two kinds called "time-interval" and "time-point." The time-interval kind corresponds to the intuitive notion of an interval of time, while the time-point kind corresponds to the notion of a point in time. The time-interval kind has three attributes: *beginning*, which returns the beginning point of an interval; *end*, which returns the end point of an interval; and *duration*, which returns the length of an interval. Time points are ordered using a primitive relation, denoted <<, that corresponds to the **before** relation. The relations between time intervals are all defined in terms of the << relation between time points. The semantics of the time-interval relations presented here are the ones found in Allen [85]. These relations, whose intuitive semantics are illustrated in Figure A-3, are all binary relations.

A.3.2 Relation Definition

In this subsection, temporal relations are formally defined.

D.26 (defrelation << (?x ?y)
:argument-type ((time-point time-point)))

D.27 (defrelation before (?x ?y)
:argument-type ((time-interval time-interval))
(forall (?x ?y) (<=> (before ?x ?y) (<< (end ?x) (beginning ?y))))

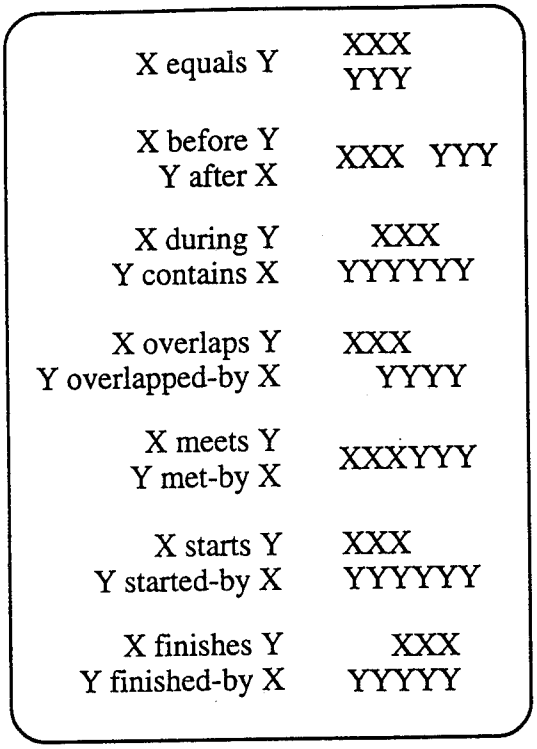


Figure A-4. Binary Temporal Relations

D.28 (defrelation after (?x ?y) := (before ?y ?x))

D.29 (defrelation during (?x ?y)
:argument-type ((time-interval time-interval))
(forall (?x ?y)
(<=> (during ?x ?y)
(and (<< (beginning ?x) (beginning ?y)) (<< (end ?y) (end ?x)))))

D.30 (defrelation contains (?x ?y) := (during ?y ?x))

D.31 (defrelation overlaps (?x ?y)
:argument-type ((time-interval time-interval))
(forall (?x ?y)
(<=> (overlaps ?x ?y)
(and (<< (beginning ?x) (beginning ?y)) (<< (end ?x) (end ?y)
(<< (beginning ?y) (end ?x)))))

D.32 (defrelation overlapped-by (?x ?y) := overlaps (?y ?x))

D.33 (defrelation meets (?x ?y)

```
        :argument-type    ((time-interval time-interval ))
        (forall (?x ?y) (<=>   (meets ?x ?y)  (= (end ?x)  (beginning ?y))))
```

D.34 (defrelation met-by (?x ?y) := meets (?y ?x))

```
D.35 (defrelation starts (?x ?y)
        :argument-type    ((time-interval time-interval ))
        (forall (?x ?y)
           (<=>  (starts ?x ?y)
                 (and   (= (beginning ?x)  (beginning ?y)) (<< (end ?x)  (end ?y)))))
```

D.36 (defrelation started-by (?x ?y) := starts (?y ?x))

```
D.37 (defrelation finishes (?x ?y)
        :argument-type    ((time-interval time-interval ))
        (forall (?x ?y)
           (<=>  (finishes ?x ?y)
                 (and   (<< (beginning ?y)  (beginning ?x)) (= (end ?x)  (end ?y)))))
```

D.38 (defrelation finished-by (?x ?y) := finishes (?y ?x))

The equals relation is the common identity relation between objects.

A.3.3 Relation Characterization

The temporal relations defined in Subsection A.3.2 are characterized in this subsection. First, the axioms that characterizes the << relation completely are presented. The properties of the temporal relations are also given. (All these properties can be deduced from the relation definitions). In the following discussion, the identity relation will not be considered a temporal relation per se. The properties of temporal relations presented here are stated for only six relations: **before, during, overlaps, meets, starts,** and **finishes.** Properties for their inverse relation can be deduced easily.

Ax.40 (forall (?x) (not (<< ?x ?x)))

Ax.41 (forall (?x ?y ?z) (=> (and (<< ?x ?y) (<< ?y ?z))

Ax.42 (forall (?x ?y) (or (<< ?x ?y) (<< ?y ?z)))

The following axioms state that all the temporal relations (excluding the identity relation) are irreflexive.

Ax.43 (forall (?x) (and (not (before ?x ?x) (not (during ?x ?x) (not (overlaps ?x ?x)
(not (meets ?x ?x)(not (starts ?x ?x) (not (finishes ?x ?x)))

The following axioms state that all the temporal relations are asymmetric. Formally stated:

Ax.44 (forall (?x ?y) (=> (equals ?x ?y) (equals ?y ?x)))

Ax.45 (forall (?x ?y) (=> (before ?x ?y) (not (before ?x ?y))))

Ax.46 (forall (?x ?y) (=> (during ?x ?y) (not (during ?x ?y))))

Ax.47 (forall (?x ?y) (=> (overlaps ?x ?y) (not (overlaps ?x ?y))))

Ax.48 (forall (?x ?y) (=> (meets ?x ?y) (not (meets ?x ?y)))

Ax.49 (forall (?x ?y) (=> (starts ?x ?y) (not (started-by ?x ?y)))

Ax.50 (forall (?x ?y) (=> (finishes ?x ?y) (not (finished-by ?x ?y)))

The following axioms state that the **before, during, starts**, and **finishes** relations are transitive.

Ax.51 (forall (?x ?y ?z) (=> (and (before ?x ?y) (before ?y ?z)) (before ?x ?z)))

Ax.52 (forall (?x ?y ?z) (=> (and (during ?x ?y) (during ?y ?z)) (during ?x ?z)))

Ax.53 (forall (?x ?y ?z) (=> (and (starts ?x ?y) (starts ?y ?z)) (starts ?x ?z)))

Ax.54 (forall (?x ?y ?z) (=> (and (finishes ?x ?y) (finishes ?y ?z)) (finishes ?x ?z)))

A.4 Spatial Relations

A.4.1 Overview

Spatial relations can be used to describe pictures (i.e., to specify the spatial relationships between objects in a picture). The IDEF5 library contains the commonly occurring spatial relations listed in Figure A-4.

left-of	far
right-of	near
above	touching
below	beside
behind	adjacent
in-front-of	disjoint
inside	intersect
outside	coincident
between	

Figure A-5. List of Spatial Relations

All listed relations are binary relations except the **between** relation, which is a ternary relation. Seventeen relations are shown in Figure A-5. The relations **right-of, below**, and **in-front-of** are the respective inverse relations of the relations **left-of, above**, and **behind**.

A.4.2 Relation Definition

In this subsection, the spatial relations listed in Figure A-5 are formally defined.

D.39 (defrelation left-of (?x ?y))

D.40 (defrelation right-of (?x ?y) := (left-of ?y ?x))

D.41 (defrelation above (?x ?y))

D.42 (defrelation below (?x ?y) := (above ?y ?x))

D.43 (defrelation behind (?x ?y))

D.44 (defrelation in-front-of (?x ?y) := (behind ?y ?x))

D.45 (defrelation near (?x ?y))

D.46 (defrelation far (?x ?y) := (near ?y ?x))

D.47 (defrelation between (?x ?y ?z))

D.48 (defrelation touching (?x ?y))

D.49 (defrelation beside (?x ?y))

D.50 (defrelation adjacent (?x ?y))

D.51 (defrelation disjoint (?x ?y))

D.52 (defrelation intersect (?x ?y))

D.53 (defrelation coincident (?x ?y))

A.4.3 Relation Characterization

In this Subsection, the spatial relations are characterized using the common relation properties of reflexivity, symmetry, antisymmetry and transitivity, as well as the relationship they bear to one another.

A.4.3.1 Reflexivity and Irreflexivity

The following axioms state that the coincident relation is reflexive. All other relations are irreflexive.

Ax.55 (forall (?x) (coincident ?x ?x))

Ax.56 (forall (?x) (not (right-of ?x ?x)))

Ax.57 (forall (?x) (not (above ?x ?x)))

Ax.58 (forall (?x) (not (behind ?x ?x)))

Ax.59 (forall (?x) (not (inside ?x ?x)))

Ax.60 (forall (?x) (not (below ?x ?x)))

Ax.61 (forall (?x) (not (in-front-of ?x ?x)))

Ax.62 (forall (?x) (not (outside ?x ?x)))

Ax.63 (forall (?x) (not (far ?x ?x)))

Ax.64 (forall (?x) (not (near ?x ?x)))

Ax.65 (forall (?x) (not (touching ?x ?x)))

Ax.66 (forall (?x) (not (beside ?x ?x)))

Ax.67 (forall (?x) (not (adjacent ?x ?x)))

Ax.68 (forall (?x) (not (disjoint ?x ?x)))

Ax.69 (forall (?x) (not (intersect ?x ?x)))

A.4.3.2 Symmetry and Antisymmetry

The following axioms state that the **near**, **far**, **across**, **faces**, and **beside** relations are symmetric, while the relations **left-of**, **above**, **behind**, and **inside** are antisymmetric.

Ax.70 (forall (?x ?y) (=> (near ?x ?y) (near ?y ?x)))

Ax.71 (forall (?x ?y) (=> (far ?x ?y) (far ?y ?x)))

Ax.72 (forall (?x ?y) (=> (beside ?x ?y) (beside ?y ?x)))

Ax.73 (forall (?x ?y) (=> (and (left-of ?x ?y) (left-of ?y ?x)) (= ?x ?y)))

Ax.74 (forall (?x ?y) (=> (and (above ?x ?y) (above ?y ?x)) (= ?x ?y)))

Ax.75 (forall (?x ?y) (=> (and (behind ?x ?y) (behind ?y ?x)) (= ?x ?y)))

Ax.76 (forall (?x ?y) (=> (and (inside ?x ?y) (inside ?y ?x)) (= ?x ?y)))

A.4.3.3 Transitivity

The following axioms state that the relations **left-of**, **right-of**, **above**, **behind**, and their inverse relations, as well as the relation **inside** are transitive.

Ax.77 (forall (?x ?y ?z) (=> (and (left-of ?x ?y) (left-of ?y ?z)) (left-of ?x ?z)))

Ax.78 (forall (?x ?y ?z) (=> (and (right-of ?x ?y) (right-of ?y ?z)) (right-of ?x ?z)))

Ax.79 (forall (?x ?y ?z) (=> (and (above ?x ?y) (above ?y ?z)) (above ?x ?z)))

Ax.80 (forall (?x ?y ?z) (=> (and (below ?x ?y) (above ?y ?z)) (below ?x ?z)))

Ax.81 (forall (?x ?y ?z) (=> (and (behind ?x ?y) (behind ?y ?z)) (behind ?x ?z)))

Ax.82 (forall (?x ?y ?z) (=> (and (in-front-of ?x ?y) (in-front-of ?y ?z)) (in-front-of ?x ?z)))

Ax.83 (forall (?x ?y ?z) (=> (and (inside ?x ?y) (inside ?y ?z)) (inside ?x ?z)))

A.4.3.4 Mutual Exclusivity

The following are sets of mutually exclusive relations: {**left-of, coincident, inside**}, {**right-of, coincident, inside**}, {**above, coincident, inside**}, {**below, coincident, inside**}, {**behind, coincident, inside**}, {**in-front-of, coincident, inside**}, {**near, coincident, inside, far**}, {**coincident, outside**}, {**beside, inside, coincident**}, {**touching, far**}, {**adjacent, far, coincident, intersect, inside**}, and {**coincident, disjoint, intersect, inside**}. The formal statements corresponding to these sets of mutually exclusive relations are organized in the following way. For each relation R, the relations that are incompatible with it (i.e., the relations in which a pair (a,b) cannot stand if (a,b) already stand in the relation R) are given.

A.84 (forall (?x ?y)
(=> (left-of ?x ?y)
(and (not (right-of ?x ?y)) (not (coincident ?x ?y)) (not (inside (?x ?y)))))

A.85 (forall (?x ?y)
(=> (right-of ?x ?y)
(and (not (left-of ?x ?y)) (not (coincident ?x ?y)) (not (inside (?x ?y)))))

A.86 (forall (?x ?y)
(=> (above ?x ?y)
(and (not (coincident ?x ?y)) (not (below ?x ?y)) (not (inside (?x ?y)))))

A.87 (forall (?x ?y)
(=> (below ?x ?y)
(and (not (above ?x ?y)) (not (coincident ?x ?y)) (not (inside (?x ?y)))))

A.88 (forall (?x ?y)
(=> (behind ?x ?y)
(and (not (in-front-of ?x ?y)) (not (coincident ?x ?y)) (not (inside (?x ?y))))))

A.89 (forall (?x ?y)
(=> (in-front-of ?x ?y)
(and (not (behind ?x ?y)) (not (coincident ?x ?y)) (not (inside (?x ?y)))))

A.90 (forall (?x ?y)
(=> (near ?x ?y)
(and (not (coincident ?x ?y)) (not (far ?x ?y)) (not (inside (?x ?y)))))

A.91 (forall (?x ?y)
(=> (far ?x ?y)
(and (not (adjacent ?x ?y)) (not (coincident ?x ?y)) (not (intersect ?x ?y))
(not (inside ?x ?y)) (not (touching ?x ?y)) (not (near ?x ?y)))))

A.92 (forall (?x ?y) (=> (outside ?x ?y)
(and (not (inside ?x ?y)) (not (coincident ?x ?y)))))

A.93 (forall (?x ?y) (=> (beside ?x ?y)
(and (not (inside ?x ?y)) (not (coincident ?x ?y)))))

A.94 (forall (?x ?y) (=> (touching ?x ?y) (not (far ?x ?y))))

A.95 (forall (?x ?y)
(=> (adjacent ?x ?y)
(and (not (far ?x ?y)) (not (coincident ?x ?y)) (not (intersect ?x ?y))
(not (inside ?x ?y)))))

A.96 (forall (?x ?y)
(=> (intersect ?x ?y)
(and (not (far ?x ?y)) (not (coincident ?x ?y)) (not (adjacent ?x ?y))
(not (disjoint ?x ?y)) (not (inside ?x ?y)))))

A.97 (forall (?x ?y)
(=> (disjoint ?x ?y)
(and (not (coincident ?x ?y)) (not (intersect ?x ?y)) (not (inside ?x ?y)))))

A.98 (forall (?x ?y)
(=> (inside ?x ?y)
(and (not (far ?x ?y)) (not (coincident ?x ?y)) (not (intersect ?x ?y))
(not (above ?x ?y)) (not (below ?x ?y)) (not (behind ?x ?y))
(not (right-of ?x ?y)) (not (near ?x ?y)) (not (beside ?x ?y))
(not (left-of ?x ?y)) (not (adjacent ?x ?y)))))

A.99 (forall (?x ?y)
(=> (coincident ?x ?y)
(and (not (left-of ?x ?y)) (not (right-of ?x ?y)) (not (above ?x ?y))
(not (below ?x ?y)) (not (behind ?x ?y)) (not (in-front-of ?x ?y))
(not (beside ?x ?y)) (not (adjacent ?x ?y)) (not (disjoint ?x ?y)))
(not (intersect ?x ?y)) (not (far ?x ?y)) (not (inside ?x ?y))
(not (outside ?x ?y)))))

A.5 Influence Relations

A.5.1 Overview

Influence relations can be used to express that an object has some effect or impact on another object. There are five influence relations, **influences** being the most general. It can be used when the type of influence or the implication of the influence between two objects is unclear or unknown. The four remaining influence relations are specializations of the general one and can be used to relate objects in some quantitative way. They are particularly appropriate in describing the influence of some measure on another. The endings in the name of the relations indicate the directional influence of one measure on another. For example, a statement of the form **(influences-pp x y)** expresses that if the measure x augments, so will the measure y. Similarly, a statement of the form **(influences-pm x y)** expresses that if the measure x augments, the measure y will decrease. A statement of the form **(influences-mp x y)** expresses that if the measure x decreases, the measure y will augment. Finally, a statement of the form **(influences-mm x y)** expresses that if the measure x decreases, so will the measure y. Figure A-6 illustrates the directional influence expressed by each relation. In the table, the first arrow indicates the behavior of the first argument and the second arrow indicates the behavior of the second argument. An arrow pointing upward indicates an increase in the value of the corresponding argument, while an arrow pointing downward indicates a decrease in the value of the corresponding argument.

Finally, an **inhibits** relation is also included. This relation can be used to express the fact that some object inhibits some other object in some undetermined way. This relation could be further specialized to describe special cases of inhibition.

Relation	Directional Influence
influences	-
influences-pp	↑↑
influences-pm	↑↓
influences-mp	↓↑
influences-mm	↓↓

Figure A-6. The Five "Influences" Relations and Their Directional Influence

A.5.2 Relation Definition

In this subsection, the influence relations are formally defined.

D.54 (defrelation influences (?x ?y))

D.55 (defrelation influences-pp (?x ?y))

D.56 (defrelation influences-pm (?x ?y))

D.57 (defrelation influences-mp (?x ?y))

D.58 (defrelation influences-mm (?x ?y))

D.59 (defrelation inhibits (?x ?y))

A.5.3 Relation Characterization

In this subsection, the influence relations are characterized by identifying their properties.

A.5.3.1 Specialization

The following axioms state that the relations **influences-pp**, **influences-pm**, **influences-mp**, and **influences-mm** are specializations of the **influences** relation.

Ax.100 (forall (?x ?y)
(=> (or (influences-pp ?x ?y) (influences-pm ?x ?y) (influences-mp ?x ?y)
(influences-mm ?x ?y))
(influences ?x ?y))

A.5.3.2 Reflexivity and Irreflexivity

The following axioms state that the relations **influences**, **influences-pp**, and **influences-pm** are reflexive. The relations **influences-pm** and **influences-mp** are irreflexive.

Ax.101 (forall (?x ?y) (influences ?x ?x))

Ax.102 (forall (?x ?y) (influences-pp ?x ?x))

Ax.103 (forall (?x ?y) (influences-mm ?x ?x))

Ax.104 (forall (?x ?y) (inhibits ?x ?x))

Ax.105 (forall (?x ?y) (not (influences-pm ?x ?x)))

Ax.106 (forall (?x ?y) (not (influences-mp ?x ?x)))

A.5.3.3 Transitivity

The following axioms state that the influence relations are transitive.

Ax.107 (forall (?x ?y ?z) (=> (and (influences ?x ?y) (influences ?y ?z))
(influences ?x ?z))

Ax.108 (forall (?x ?y ?z) (=> (and (influences-pp ?x ?y) (influences-pp ?y ?z))
(influences-pp ?x ?z))

Ax.109 (forall (?x ?y ?z)
(=> (and (influences-pm ?x ?y) (influences-pm ?y ?z))
(influences-pm ?x ?z))

Ax.110 (forall (?x ?y ?z)
(=> (and (influences-mp ?x ?y) (influences-mp ?y ?z))
(influences-mp ?x ?z))

Ax.111 (forall (?x ?y ?z)
(=> (and (influences-mm ?x ?y) (influences-mm ?y ?z))
(influences-mm ?x ?z))

Ax.112 (forall (?x ?y ?z) (=> (and (inhibits ?x ?y) (inhibits ?y ?z)) (inhibits ?x ?z))

A.5.3.4 Mutual exclusivity

The following axioms state that the specialized influence relations are mutually exclusive.

Ax.113 (forall (?x ?y)
(=> (influences-pp ?x ?y)
(and (not (influences-pm ?x ?y)) (not (influences-mp ?x ?y))
(not (influences-mm ?x ?y)))

Ax.114 (forall (?x ?y)
(=> (influences-pm ?x ?y)
(and (not (influences-pp ?x ?y)) (not (influences-mp ?x ?y))
(not (influences-mm ?x ?y)))

Ax.115 (forall (?x ?y)
(=> (influences-mp ?x ?y)
(and (not (influences-pm ?x ?y)) (not (influences-pp ?x ?y))
(not (influences-mm ?x ?y)))

Ax.116 (forall (?x ?y)
(=> (influences-mm ?x ?y)
(and (not (influences-pm ?x ?y)) (not (influences-mp ?x ?y))
(not (influences-pp ?x ?y)))

A.6 Dependency Relations

A.6.1 Overview

The dependency relations can be used to express the fact that an object depends on another. The dependency can be general (in which case the **"depends-on"** relation is used) or may be more specific. In particular, the dependency may be existential (in which case the **"depends-on-existentially"** relation is used) or causal (in which case the **"depends-on-causally"** relation can

be used). An object **A** is existentially dependent on an object **B** if the existence of **A** depends on the existence of **B**. In such a case, if **B** ceases to exist, then **A** will also cease to exist. An object **A** is causally dependent on an object **B** if **A** is the result or effect of the existence of **B**. In other words, **B** is the cause or one of the causes for the existence of **A**. The **existentially-dependent** and **causally-dependent** relations differ in that, with the latter relation, **B**'s destruction may not cause **A** to cease to exist.

A.6.2 Relation Definition

In this subsection, the dependency relations are formally defined. To enable a detailed definition of the **depends-on-existentially** relation, an **exists** relation that takes one argument is also defined. A statement of the form **(exists A)** denotes the fact that **A** exists.

D.60 (defrelation exists (?x)

D.61 (defrelation depends-on (?x ?y))

D.62 (defrelation depends-on-existentially (?x ?y)
(forall (?x ?y)
(=> (and (depends-on-existentially ?x ?y) (not (exists ?y))) (not (exists ?x)))))

D.63 (defrelation depends-on-causally (?x ?y))

A.6.3 Relation Characterization

In this subsection, the dependency relations are charcterized by identification of their properties.

A.6.3.1 Specialization

The following axioms state that the relations **depends-on-existentially** and **depends-on-causally** are specializations of the relation depends-on.

Ax.117 (forall (?x ?y)
(=> (or (depends-on-existentially ?x ?y)
(depends-on-causally ?x ?y))
(depends-on ?x ?y))

A.6.3.2 Transitivity

The following axioms state that the three dependency relations are transitive.

```
Ax.118  (forall (?x ?y ?z)
            (=>  ( and (depends-on ?x ?y) (depends-on ?y ?z))
                 (depends-on ?x ?z))

Ax.119  (forall (?x ?y ?z)
            (=>  ( and  (depends-on-existentially  ?x ?y)
                        (depends-on-existentially  ?y ?z))
                 (depends-on-existentially  ?x ?z))

Ax.120  (forall (?x ?y ?z)
            (=>  ( and (depends-on-causally  ?x ?y) (depends-on-causally  ?y ?z))
            (depends-on-causally  ?x ?z))
```

A.7 Case Relations

A.7.1 Overview

The case relations differ from other families of relations introduced so far in that they do not depend solely on the nature or the meaning of the terms they relate [Winston, et. al. 87]. Each case relation provides a knowledge structure without which the relation cannot exist. Five case relations that are common in that they all can be used to describe parts of an event are defined and characterized. An event typically involves an agent, an action, an instrument or an object, and a recipient. The five case relations presented here involved two such components of an event. The five relations are as follows.

- **Agent-Action** This relation relates an agent of an event to the corresponding action. The arguments to this relation should be a kind and a verb. Examples of terms related through this relation are **dog** and **bark**, and **artist** and **paint**.

- **Agent-Instrument** This relation relates an agent of an event to the corresponding instrument. Examples of the use of this relation are **A skier uses skis** and **A soldier uses a gun**.

- **Agent-Object** This relation relates an agent of an event to the object that takes part in the event. This relation is used in case the object is neither the recipient nor the instrument. Examples of terms related through such relations are **writer** and **paper** and **baker** and **flour**.

- **Action-Recipient** This relation relates the recipient of an event to the corresponding action. Examples of terms related through this relation are **lay down** and **bed** and **type** and **keyboard**.

- **Action-Instrument** This relation relates the instrument in an event to the corresponding action. Examples of terms related through this relation are **paint** and **brush** and **strum** and **guitar**.

A.7.2 Relation Definition

In this subsection, the case relations are formally defined.

D.64 (defrelation agent-action (?x ?y))

D.65 (defrelation agent-instrument (?x ?y))

D.66 (defrelation agent-object (?x ?y))

D.67 (defrelation agent-recipient (?x ?y))

D.68 (defrelation action-instrument (?x ?y))

A.7.3 Relation Characterization

In this subsection, the case relations are charaterized

A.7.3.1 Irreflexivity

The following axioms state that the five case relations are irreflexive.

Ax.121 (forall (?x) (not (agent-action ?x ?x))

Ax.122 (forall (?x) (not (agent-instrument ?x ?x))

Ax.123 (forall (?x) (not (agent-object ?x ?x))

Ax.124 (forall (?x) (not (agent-recipient ?x ?x))

Ax.125 (forall (?x) (not (action-instrument ?x ?x))

A.7.3.2 Asymmetry

The following axioms state that the five case relations are asymmetric.

Ax.126 (forall (?x ?) (=> (agent-action ?x ?y) (not (agent-action ?y ?x))))

Ax.127 (forall (?x ?) (=> (agent-instrument ?x ?y) (not (agent-instrument ?y ?x))))

Ax.128 (forall (?x ?) (=> (agent-object ?x ?y) (not (agent-object ?y ?x))))

Ax.129 (forall (?x ?) (=> (agent-recipient ?x ?y) (not (agent-recipient ?y ?x))))

Ax.130 (forall (?x ?) (=> (action-instrument ?x ?y) (not (action-instrument ?y ?x))))

Appendix B. Grammar for the IDEF5 Elaboration Language

B.1 Constant and variables

<word> ::= *a primitive syntactic object*
<expression> ::= <word> | (<expression>*)

<indvar> ::= *a word beginning with the character* ?
<predvar> ::= *a word beginning with the character* #
<variable> ::= <indvar> | <predvar>

<I5-ontology-constant>::= *a word denoting an ontology*
<logical-constant> ::= *a word denoting a truth value* | <ontology-constant>::<logical-constant>
<I5-individual-constant> ::= *a word denoting an individual*
| <ontology-constant>::<I5-individual-constant>
<I5-kind-constant> ::= *a word denoting a kind* | <ontology-constant>::<I5-kind-constant>
<I5-SO-pred-constant> ::= *a word denoting a second order predicate*
| <ontology-constant>::<I5-SO-pred-constant>
<I5-TO-pred-constant> ::= *a word denoting a third order predicate*
| <ontology-constant>::<I5-TO-pred-constant>
<relation-constant> ::= <I5-SO-pred-constant> | <I5-TO-pred-constant>
<I5-function-constant> ::= *a word denoting a function*
| <ontology-constant>::<I5-function-constant>
<I5-attribute-constant> ::= *a word denoting an attribute*
| <ontology-constant>::<I5--attribute-constant>
<I5-property-constant> ::= *a word denoting a property*
| <ontology-constant>::<I5-property-constant>
<I5-source-statement-constant> ::= *a word denoting a statement*
| <ontology-constant>::<I5-source-statement-constant>
<I5-ontology-term-constant> ::= *a word denoting a term*
| <ontology-constant>::<I5-ontology-term-constant>
<I5-source-constant> ::= *a word denoting a source*
| <ontology-constant>::<I5-source-constant>
<I5-note-constant> ::= *a word denoting a note*
| <ontology-constant>::<I5-note-constant>
<I5-object-state-constant> ::= <I5-kind-constant>:<I5-property-constant> |

<ontology-constant>::<I5-kind-constant>:<I5-property-constant>

<I5-constant> ::= <I5-property-constant> | <I5-SO-pred-constant> | <I5-TO-pred-constant> |
<I5-function-constant> | <I5-kind-constant> | <I5-ontology-term-constant>
| <I5-source-constant> | <I5-note-constant> | <I5-source-statement-constant>
| <I5-attribute-constant> | <I5-individual-constant>

<constant> ::= <I5-constant> | <I5-ontology-constant> | <logical-constant>

B.2 Operators

<definition-operator> ::= define-relation | define-function | define-individual | := | :argument-type

<term-operator> ::= listof | setof | if | cond | the | setofall

<sentence-operator> ::= = | /= | not | and | or | implies | equiv | forall | exists | nec | poss | <I5-sentence-operator>

<I5-sentence-operator> := <I5-kind-operator> | <I5-ontology-operator> | <I5-note-operator> | <I5-relation-operator> | <I5-function-operator> | <I5-individual-operator> | <I5-property-operator> | <I5-attribute-operator> | <I5-statement-operator> | <I5-ontology-term-operator> | <I5-source-operator> | I5-process

<I5-kind-operator>::= I5-kind | I5-kind-property | I5-kind-attribute | I5-has-synonyms | I5-kind-description | I5-referenced-relations | defining | essential | subkind-of | I5-object-state

<I5-ontology-operator> ::= I5-ontology | I5-ontology-context | I5-ontology-viewpoint | I5-ontology-purpose | I5-ontology-project | I5-ontology-analyst | I5-ontology-reviewer | I5-ontology-description

<I5-individual-operator> ::= I5-individual | I5-individual-description | I5-instance-of

<I5-relation-operator> ::= I5-relation | I5-relation-description | I5-relation-arity | I5-rel-arg-type

<I5-function-operator> ::= I5-function | I5-function-description | I5-function-arity | I5-fct-arg-type

<I5-property-operator> ::= I5-property | I5-property-description | has-property

<I5-attribute-operator> ::= I5-attribute | I5-attribute-description | I5-attribute-type | I5-attribute-applies-to

<I5-source-statement-operator> ::= I5-source-statement | I5-source-statement-description | I5-source-statement | I5-has-original-statement | I5-status-type | active_original | active_derived | retired_original | retired_derived

<I5-ontology-term-operator> ::= I5-ontology-term | I5-use-statements | I5-sources-used | I5-ontology-term-description | I5-ontology-term-description

<I5-note-operator> ::= I5-note | I5-note-description | I5-has-note

<I5-source-operator> ::= I5-source | I5-source-description | I5-collected-from | I5-collected-by | I5-source-abstract | I5-source-purpose | I5-support-ontology-terms | I5-support-statement | I5-has-supporting-sources

<operator> ::= <definition-operator> | <term-operator> | <sentence-operator>

B.3 Terms

<term> ::= <indvar> | <I5-constant> | <I5-ontology-constant> | <funterm> | <listterm> |
<setterm> | <logterm> | quanterm> | <I5-attribute-term>
<funterm> ::= (<I5-function-constant> <term>$^{+}$)
<I5-attribute-term> ::= (<I5-attribute-constant> <individual-constant> | <indvar>) |
<setterm> ::= (setof <term>*)
<listterm> ::= (listof <term>*)
<logterm> ::= (if <sentence> <term> [<term>]) |
(cond (<sentence> <term>) (<sentence> <term>)$^{+}$)
<quanterm> ::= (the <term> <sentence>) |
(setofall <term> <sentence>)

B.4 Definitions

<definition> ::= <partial-definition> | <complete-definition>

<complete-definition> ::= (define-individual <individual-constant> := <term>) |
(define-function <function-constant> ({<indvar> | <predvar>}$^{+}$)
[:argument-type (({<I5-kind-constant> |
<I5-object-state-term>}$^{+}$)$^{+}$)]
:= <term>) |
(define-relation <relation-constant> ({<indvar> | <predvar>}$^{+}$)
[:argument-type (({<I5-kind-constant> |
<I5-object-state-term>}$^{+}$)$^{+}$)]
:= <sentence>)

<partial-definition> ::= (define-individual <individual-constant> [<sentence>]) |
(define-function <function-constant> ({<indvar> | <predvar>}$^{+}$)
[:argument-type (({<I5-kind-constant> |
<I5-object-state-term>}$^{+}$)$^{+}$)]
[<sentence>])) |
(define-relation <relation-constant> ({<indvar> | <predvar>}$^{+}$)
[:argument-type (({<I5-kind-constant> |
<I5-object-state-term>}$^{+}$)$^{+}$)]
[<sentence>]))

B.5 Sentences

B.5.1 General Sentences

<sentence> ::= <logical-constant> | <equation> | <inequality> | <relsent> | <logsent>
<quantsent> | <I5-sentence>

<equation> ::= (= <term> <term>)

<inequality> ::= (/= <term> <term>)

<relsent> ::= (<relation-constant> <term>$^{+}$) | (<function-constant> <term> <term>*)

<logsent> ::= (not <sentence>) | (and <sentence> <sentence>+) | (poss <sentence>) |
(implies <sentence> <sentence>) | (equiv <sentence> <sentence>) |
(nec <sentence>)

<quantsent> ::= (forall ([<indvar> | <predvar>]+) <sentence>) |
(exists ([<indvar> | <predvar>]+) <sentence>)

B.5.2 IDEF5 Specific Sentences

<I5-sentence> ::= <I5-kind-sentence> | <I5-ontology-sentence> | <I5-individual-sentence> |
<I5-property-sentence> | <I5-attribute-sentence> | <I5-statement-sentence> |
<I5-ontology-term-sentence> | <I5-source-sentence> | <I5-note-sentence> |
<I5-relation-sentence> | <I5-function-sentence> | <I5-process-decl>

B.5.2.1 IDEF5 Ontology Sentences

<I5-ontology-sentence>::= <I5-ontology-decl> | <I5-ontology-context-decl> |
<I5-ontology-viewpoint-decl> | <I5-ontology-purpose-decl> |
<I5-ontology-project-decl> | <I5-ontology-analyst-decl> |
<I5-ontology-reviewer-decl> | <I5-ontology-description-decl>
<I5-ontology-decl> ::= (I5-ontology <I5-ontology-constant> | <indvar>)
<I5-ontology-context-decl> ::= (I5-ontology-context <I5-ontology-constant> | <indvar>
string)
<I5-ontology-viewpoint-decl> ::= (I5-ontology-viewpoint <I5-ontology-constant> | <indvar>
string)
<I5-ontology-purpose-decl> ::= (I5-ontology-purpose <I5-ontology-constant> | <indvar>
string)
<I5-ontology-project-decl> ::= (I5-ontology-project <I5-ontology-constant> | <indvar> string)
<I5-ontology-analyst-decl> ::= (I5-ontology-analyst <I5-ontology-constant> | <indvar> string)
<I5-ontology-reviewer-decl> ::= (I5-ontology-rewiever <I5-ontology-constant> | <indvar>
string)
<I5-ontology-description-decl> ::= (I5-ontology-description <I5-ontology-constant> |
<indvar> string)
<I5-in-ontology-decl> ::= (in-ontology <I5-constant> | <predvar>
<I5-ontology-constant> | <indvar>)

B.5.2.2 IDEF5 Kind Sentences

<I5-kind-sentence> := <I5-kind-decl> | <I5-kind-property-decl> | <I5-kind-attribute-decl> |
<I5-kind-description-decl> | <I5-has-synonyms-decl> |
<I5-referenced-relations-decl> | <I5-object-state-decl>
<I5-kind-decl> ::= (I5-kind <I5-kind-constant> | <predvar>)
<I5-kind-property-decl> ::= (I5-kind-property <I5-kind-constant> | <predvar>
<I5-property-constant> | <predvar> [defining] [essential])
<I5-kind-attribute-decl> ::= (I5-kind-attribute <I5-kind-constant>| <predvar> |
<I5-attribute-constant> | <predvar>)
<I5-kind-description-decl> ::= (I5-kind-description <I5-kind-constant> | <predvar> string)
<I5-kind-synonyms-decl> ::= (I5-kind-synonyms <I5-kind-constant> | <predvar>
({<I5-ontology-term> | <indvar>}+))
<I5-referenced-relations-decl> ::= (I5-referenced-relations <I5-kind-constant> | <predvar>
({<I5-relation-constant> | <predvar>}+))

<I5-subkind-of-decl> ::= (I5-subkind-of <I5-kind-constant> | <predvar>
<I5-kind-constant> | <predvar>)
<I5-object-state-decl> ::= (I5-object-state <I5-object-state-constant> | <predvar>)

B.5.2.3 IDEF5 Individual Sentences

<I5-individual-sentence> ::= <I5-individual-decl> | <I5-individual-description-decl> |
<I5-instance-of-decl>
<I5-individual-decl> ::= (I5-individual <I5-individual-constant> | <indvar>)
<I5-individual-description-decl> ::= (I5-individual-description <I5-individual-constant> |
<indvar> string)
<I5-is-of-kind-decl> ::= (I5-is-of-kind <I5-individual-constant> | <indvar>
<I5-kind-constant> | <predvar>)

B.5.2.4 IDEF5 Property Sentences

<I5-property-sentence> ::= <I5-property-decl> | <I5-property-description-decl> |
<has-property-decl>
<I5-property-decl> ::= (I5-property <I5-property-constant> | <predvar>)
<I5-property-description-decl> ::= (I5-property-description <I5-attribute-constant> |
<predvar> string)
<I5-has-property-decl> ::= (I5-has-property <I5-individual-constant> | <predvar>
<I5-property-constant> | <predvar>)

B.5.2.5 IDEF5 Attribute Sentences

<I5-attribute-sentence> ::= <I5-attribute-decl> | <I5-attribute-description-decl> |
<I5-attribute-type-decl> |<I5-attribute-applies-to-decl>
<I5-attribute-decl> ::= (I5-attribute <I5-attribute-constant> | <predvar> <I5-attribute-type>)
<I5-attribute-type-decl> ::= <list-type> | <I5-kind-constant> | <set-type>
<list-type> ::= (listof <I5-kind-constant> | (or <I5-kind-constant> <I5-kind-constant>$^{+}$)) |
(list-of <list-type>) | (listof <type>)
<I5-attribute-description-decl> ::= (I5-attribute-description <I5-attribute-constant> |
<predvar> string)
<I5-attribute-applies-to -decl> ::= (I5-attribute-applies-to <I5-attribute-constant> | <predvar>
<I5-individual-constant> | <indvar>)

B.5.2.6 IDEF5 Relation Sentences

<I5-relation-sentence> ::= <I5-relation-decl> | <I5-relation-description-decl> |
<I5-rel-arg-type-decl> | <I5-relation-arity-decl>
<I5-relation-decl> ::= (I5-relation <predvar> | <I5-relation-constant>)
<I5-relation-arity-decl> ::= (I5-relation-arity <relation-constant> | <predvar>
pos-int | <indvar>)
<I5-rel-arg-type-decl> ::= (I5-rel-arg-type <relation-constant> | <predvar>
(({<I5-kind-constant> | <predvar> | <I5-object-state-term>}$^{+}$)$^{+}$))
<I5-relation-description-decl> ::= (I5-relation-description <I5-relation-constant> | <predvar>
string)

B.5.2.7 IDEF5 Function Sentences

```
<I5-function-sentence> ::=  <I5-function-decl> | <I5-function-description-decl> |
                            <I5-function-arity-decl> | <I5-fct-arg-type-decl>
<I5-function-decl> ::=  (I5-function <I5-function-constant> | <predvar>)
<I5-function-arity-decl> ::=  (I5-function-arity <I5-function-constant> | <predvar>
                                pos-int | <indvar>)
<I5-fct-arg-type-decl> ::=  (I5-fct-arg-type <I5-function-constant> | <predvar>
                              (({<I5-kind-constant> | <predvar> | <I5-object-state-term>}+)+))
<I5-function-description-decl> ::=  (I5-function-description <I5-function-constant> |
                                      <predvar> string)
```

B.5.2.8 IDEF5 Source Sentences

```
<I5-source-sentence> ::= <I5-source-decl> | <I5-source-description-decl> |
                         <I5-collected-from-decl> | <I5-collected-by-decl> |
                         <I5-source-abstract-decl> | <I5-source-purpose-decl> |
                         <I5-support-ontology-terms-decl> | <I5-support-statement-decl> |
                         <I5-has-supporting-sources-decl>

<I5-source-decl> ::=  (I5-source <source-constant> | <indvar>)
<I5-source-description-decl> ::=  (I5-source-description <I5-source-constant> | <indvar> string)
<I5-collected-from-decl> ::=  (I5-collected-from <I5-source-constant> | <indvar> string)
<I5-collected-by-decl> ::=  (I5-collected-by <I5-source-constant> | <indvar>string)
<I5-source-abstract-decl> ::=  (I5-source-abstract <I5-source-constant> | <indvar> string)
<I5-source-purpose-decl> ::=  (I5-source-purpose <I5-source-constant> | <indvar> string)
<I5-support-ontology-terms-decl> ::=  (I5-support-ontology-terms <I5-source-constant> |
                                        <indvar> ({<I5-term-constant> | <indvar>}+))
<I5-support-statement-decl> ::=  (I5-support-statements <I5-source-constant> | <indvar>
                                   ({<I5-source-statement-constant> | <indvar>}+) )
<I5-has-supporting-sources-decl> ::=  (I5-has-supporting-sources <I5-constant> |
                                        <indvar> | <predvar>
                                        ({<I5-source-constant> | <indvar>}+))
```

B.5.2.9 IDEF5 Source-Statement Sentences

```
<I5-source-statement-sentence> ::=  <I5-source-statement-decl> | <I5-status-decl> |
                                    <I5-source-statement-description-decl> |
                                    <I5-source-statement-decl> |
                                    <I5-has-original-statement-decl> |
                                    <I5-has-supporting-sources-decl>

<I5-source-statement-decl> ::= (I5-source-statement <I5-source-statement-constant> | <indvar>)
<I5-source-statement-description-decl> ::=  (I5-source-statement-description
                                              <I5-source-statement-constant> | <indvar> string)
<I5-status-decl> ::=  (I5-status <I5-source-statement-constant>| <indvar>
                        <I5-status-type-decl>
<I5-status-type-decl> ::= active_original | active_derived |  retired_original | retired_derived
<I5-has-original-statement-decl> ::=  (I5-has-original-statement
                                        <I5-source-statement-constant> | <indvar>
                                        <I5-source-statement-constant> | <indvar>)
```

B.5.2.10 IDEF5 Ontology-Term Sentences

<I5-ontology-term-sentence> ::= <I5-ontology-term-decl> | <I5-use-statements-decl> |
<I5-ontology-term-description-decl> |
<I5-ontology-term-description-decl>

<I5-ontology-term-decl> ::= (I5-ontology-term <I5-term-constant> | <indvar>)
<I5-ontology-term-description-decl> ::= (I5-ontology-term-description <I5-term-constant> |
<indvar> string)
<I5-uses-statements-decl>::= (use-statements <I5-ontology-term-constant> | <indvar>
({<I5-source-statement-constant> | <indvar>}$^{+}$))

B.5.2.11 IDEF5 Note Sentences

<I5-note-sentence> ::= <I5-note-decl> | <I5-note-description-decl> | <I5-has-note-decl>

<I5-note-decl> ::= (I5-note <I5-note-constant> | <indvar>)
<I5-note-description-decl> ::= (I5-note-description <I5-note-constant> | <indvar> string)
<I5-has-note-decl> ::= (I5-has-note <I5-constant> | <indvar> | <predvar> <I5-note-constant>
| <indvar>)

B.5.2.12 Object State Related Constructs and Relations

<I5-process-decl> ::= (I5-process <I5-process-constant> | <predvar>)

The following relations are predefined and chracterized as follows:

```
(define-relation part-of (?x ?y))
(forall (#x #y) (=> (I5-rel-arg-type part-of (#x #y))
                    (and (I5-kind #x) (I5-kind #y))))

(define-relation transitions-to (?x ?y))
(forall (#x #y) (=> (I5-rel-arg-type transitions-to (#x #y))
                    (and (or (I5-kind #x) (I5-object-state #x))
                         (or (I5-kind #y) (I5-object-state #y)))))

(define-relation inst-transitions-to (?x ?y))
(forall (#x #y) (=> (I5-rel-arg-type transitions-to (#x #y))
                    (and (or (I5-kind #x) (I5-object-state #x))
                         (or (I5-kind #y) (I5-object-state #y)))))

(define-relation transitions-during (?x ?y ?z))
(forall (#x #y #z) (=> (I5-rel-arg-type transitions-to (#x #y))
                       (and (or (I5-kind #x) (I5-object-state #x))
                            (or (I5-kind #y) (I5-object-state #y))
                            (I5-process #z))))

(define-relation inst-transitions-during (?x ?y ?z))
(forall (#x #y #z) (=> (I5-rel-arg-type transitions-to (#x #y))
                       (and (or (I5-kind #x) (I5-object-state #x))
                            (or (I5-kind #y) (I5-object-state #y))
                            (I5-process #z))))
```

```
(define-relation s-transitions-to (?x ?y ))
(forall (#x #y) (=>  (I5-rel-arg-type transitions-to  (#x #y))
                     (and  (or (I5-kind #x) (I5-object-state #x))
                           (or (I5-kind #y) (I5-object-state #y)))))

(define-relation inst-s-transitions-to (?x ?y ))
(forall (#x #y) (=>  (I5-rel-arg-type transitions-to  (#x #y))
                     (and  (or (I5-kind #x) (I5-object-state #x))
                           (or (I5-kind #y) (I5-object-state #y)))))

(define-relation s-transitions-during (?x ?y ?z))
(forall (#x #y #z) (=>  (I5-rel-arg-type transitions-to  (#x #y))
                        (and  (or (I5-kind #x) (I5-object-state #x))
                              (or (I5-kind #y) (I5-object-state #y))
                              (I5-process #z))))

(define-relation inst-s-transitions-during (?x ?y ?z))
(forall (#x #y #z) (=>  (I5-rel-arg-type transitions-to  (#x #y))
                        (and  (or (I5-kind #x) (I5-object-state #x))
                              (or (I5-kind #y) (I5-object-state #y))
                              (I5-process #z))))
```

Bibliography

Allen, J. F. (1984). "Towards A General Theory Of Action And Time." *Artificial Intelligence* 23: 123-154.

Anupindi, S. R. (1992). Semantic Requirements for an Integrated Bill of Materials System. M.S. Thesis. Dept. of Industrial Engineering, Texas A&M University, College Station, TX.

Aczel, P., Israel, D., Katagiri, Y., and Peters, S. (Eds.) (1993). Situation Theory and its Applications, Volume 3. CSLI Lecture Notes 37.

Barwise, J., Gawron, M., Plotkin, G., and Tutiya, S. (Eds.) (1991). Situation Theory and its Applications, Volume 2. CSLI Lecture Notes 26.

Bealer, G. (1980). Quality and Concept. Oxford: Oxford University Press.

Barwise, J. and Perry, J. (1983). Situations and Attitudes. Cambridge: MIT Press.

Benjamin, P. C., Menzel, C., and Mayer, R. J. (forthcoming). "Towards a Method for Acquiring CIM Ontologies." To appear in *International Journal of CIM*. (Expected 1994).

Brachman, R. J. (1983). "What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks." *IEEE Computer*. October.

Burkhart, R., et al. (1991). IRDS Conceptual Schema Working Paper. ISO/IEC JTC1/SC21/WG3 N. ANSI X3H3 Working Committee.

Chaffin, R. and Herrmann, D. J., (1987). "Relation Element Theory: A New Account of the Representation and Processing of Semantic Relations." In Memory and Learning: The Ebbinghaus Centennial Conference. Gorfein, D. S. (Ed.). Hillsdale, NJ: Erlbaum.

Chen, P. (1976). "The Entity-Relationship Model: Toward a Unified View of Data." *ACM Transactions on Database Systems* 1(1): 9-36.

Cohen, N. J. (1987). "Preserved Learning Capacity in Amnesia: Evidence for Multiple Memory Systems." Neuropsychology of Memory. New York: Guilford Press, 1987: 83-103.

Coleman, D. S. (1989). A Framework for Characterizing the Methods and Tools of an Integrated System Engineering Methodology (ISEM), Draft 2 Rev. 0. Santa Monica, CA: Pacific Information Management, Inc.

Devlin, K. (1991). Logic and Information. Cambridge: Cambridge University Press.

Enderton, H. (1972). A Mathematical Introduction to Logic. New York, Academic Press.

Fulton J, et al. (1991). "The Semantic Unification Meta-model: Technical Approach." Draft Report of the Dictionary/Methodology Committee of IGES/PDES. Version 0, Release 6, Draft 3.

Futrell, M. T. (1991). The IDEF5 Application Procedure. Master's Project Report. Department of Industrial Engineering, Texas A&M University, College Station, TX.

Genesereth, M. R. and Fikes, R. E. (1992). Knowledge Interchange Format Version 3.0 - Reference Manual. Report Logic-92-1. Logic Group, Stanford University, CA.

Gruber, T. R. (1992). Ontolingua: A Mechanism to Support Portable Ontologies, Knowledge Systems Laboratory Technical Report KSL 91-66, Final Version. Stanford University.

Gruber, T. R. (1993). "A translation approach to portable ontologies." *Knowledge Acquisition*, 5(2):199-220, 1993.

Guha, R. V. and Lenat, D. V. (1990). "CYC: A Mid-Term Report." *AI Magazine* 11(3): 32-59.

Hobbs, J. R., et al. (1987). "Commonsense Metaphysics and Lexical Semantics." *Computational Linguistics* 13(3-4): 241-250.

Hobbs, J., Croft, W., Davies, T., Edwards, D., and Laws, K. (1987). The TACITUS Commonsense Knowledge Base. Artificial Intelligence Research Center, SRI International.

Information Processing Systems: Concepts and Terminology for the Conceptual Schema and the Information Base. (ISO/TR 9007). (1987). International Standards Organization.

Integrated Computer-Aided Manufacturing (ICAM) Architecture. Pt. II Vol. IV: Function Modeling Manual (IDEFØ) (DTIC-B062457). (1981). SofTech, Inc.

Integrated Computer-Aided Manufacturing (ICAM) Architecture. Pt. II, Vol. V: Information Modeling Manual (IDEF1) (DTIC-B062457). (1981). SofTech, Inc.

Integrated Computer-Aided Manufacturing (ICAM) Architecture. Pt. II, Vol. VI: Dynamics Modeling Manual (IDEF2) (DTIC-B062457). (1981). SofTech, Inc.

Integrated Information Support System (IISS), Volume 5: Common Data Model Subsystem: Part 4: Information Modeling Manual. (DTIC-A181952). (1985). General Electric.

International Standards Organization (1987). Concepts and Terminology for the Conceptual Schema. ISO Technical Report TR9007.

Jackson, P. (1990). Introduction to Expert Systems. Addison-Wesley, 1990.

Knowledge Based Systems, Inc. (KBSI) (1991). Formal Foundations for an Ontology Description Method. Technical Report, KBSI-SBONT-91-TR-01-1291-02.

Knowledge Based Systems, Inc. (KBSI) (1992). Situation Based Ontology: Phase I Report. DARPA SBIR Contract No. DAAH01-91-C-R236.

Knowledge Based Systems, Inc. (KBSI) (1993). Ontology-Driven Information Integration: Phase I Final Report. NASA SBIR Project, Contract No. NAS-9-18829.

Kripke, S. (1963). "Semantical Considerations on Modal Logic." *Acta Philosophica Fennica* 16: 39-48.

Lenat, D., Prakash, M., and Shepherd, M. (1986). "CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks.: *The AI Magazine*. Summer.

Link, G. (1983). "The Logical Analysis of Plurals and Mass Terms: A Lattice Theoretic Approach." In R. Bauerle (Ed.), Meaning, Use, and Interpretation. Berlin: De Gruyter.

Loiselle, C. L. and Cohen, P. L. (1989). "Explorations in the Contributors to Plausibility." COINS Technical Report 89-29. University of Massachusetts, Amherst, MA.

Mayer, R. J., et al. (1987). Knowledge-based Integrated Information Systems Development Methodologies Plan, Volume 2. (DTIC-A195851).

Mayer, R. J. (Ed.). (1990). IDEFØ Function Modeling: A Reconstruction of the Original Air Force Report. College Station, TX: Knowledge Based Systems, Inc.

Mayer, R. J. (Ed.). (1990). IDEF1 Information Modeling: A Reconstruction of the Original Air Force Report. College Station, TX: Knowledge Based Systems, Inc.

Mayer, R. J. (Ed.). (1990). IDEF1X Data Modeling: A Reconstruction of the Original Air Force Report. College Station, TX: Knowledge Based Systems, Inc.

Mayer, R. J., Menzel, C. P., and deWitte, P. S (1991). IDEF3 Technical Report. WPAFB, OH: AL/HRGA.

Mayer, R. J., Menzel, C. P, and Mayer, P. S. D. (1991). IDEF3: A Methodology for Process Description. WPAFB, OH: AL/HRGA.

Mayer, R. J., Edwards, D. A., Decker, L. P., and Ackley, K. A. (1991). IDEF4 Technical Report. WPAFB, OH: Al/HRGA.

Mayer, R. J., deWitte, P., Griffith, P., and Menzel, C. P. (1991). IDEF6 Concept Report. WPAFB, OH: AL/HRGA.

Mayer, R. J. and deWitte, P. (1991). Framework Research Report. WPAFB, OH: AL/HRGA.

Mayer, R. J., Menzel, C., Painter, M., and Benjamin, P. C. (1993). "The Role of Ontology in Enterprise Integration." In Proceedings of the May 1993 IDEF Users Group Conference. College Park, MD.

Mayer, R. J., Benjamin, P. C., Caraway, B. E., and Painter, M. K. (forthcoming). "A Framework and a Suite of Methods for Business Process Reengineering." In Business Process Reengineering: A Managerial Perspective. Kettinger, B. and Grover, V. (Eds.) (Expected 1994).

McGraw, K. and Briggs, K. (1989). Knowledge Acqusition Principles and Guidelines. Prentice Hall.

Menzel, C. (1990). "Actualism, Ontological Commitment, and Possible World Semantics" *Synthese* 85: 355-389.

Menzel, C. (1991). "The True Modal Logic." *Journal of Philosophical Logic* 20: 331-374.

Menzel, C. and Mayer, R. J. (1991). "Theoretical Foundations for Information Representation and Constraint Specification." Technical Paper #AL-TP-1991-0044. Human Resources Directorate, Logistics Research Division, WPAFB, OH.

Menzel, C., Mayer, R., and Sanders, L. (1992). "Representation, Information Flow, and Model Integration." In [Petrie 92], 131-141.

Menzel, C., Mayer, R., and Edwards, D. (1994). "IDEF3 Process Descriptions and Their Semantics." In Kuziak, A., and Dagli, C. Intelligent Systems in Design and Manufacturing. ASME Press.

The Merriam-Webster Dictionary. (1986) New York: Simon and Schuster.

Musen, M. A. (1989). "Conceptual Models of Interactive Knowledge-Acquisition Tools." *Knowledge Acquisition* 1. 73-88.

Neches, R., et al. (1991). "Enabling Technology for Knowledge Sharing." *AI Magazine* 12(3): 36-56.

Painter, M. K. (1990). "Modeling with an IDEF Perspective: Some Practical Insights." Proceedings, SME Autofact 90. Dearborn, MI: Society of Manufacturing Engineers.

Painter, M. K. (1991). "Information Integration for Concurrent Engineering (IICE): Program Foundations and Philosophy." IDEF Users Group Conference Proceedings. May.

Petrie, C. (1992). Enterprise Integration Modeling. Proceedings of the First International Conference. Cambridge: MIT Press.

Ross, D. T. (1985). "SADT Today: A Retrospective on an Idea." *IEEE Computer Magazine* (special issue on Requirements Engineering).

Sarris, A. K. (1992). "Needs Analysis and Requirements Document: Integration Toolkit and Methods, Corporate Data Integration Tools." MANTECH Report WL-TR-92-8027. WPAFB, OH.

Sowa, J. (1984). Conceptual Structures: Information Processing in Mind and Machine. Reading, MA: Addison Wesley.

Tarski, A. (1983). "The Concept of Truth in Formalized Languages." In Logic, Semantics, and Metamathematics. Oxford: Oxford University Press.

Touretzky, D. S. (1984). "Implicit Ordering of Defaults in Inheritance Systems." Proceedings of the 5th National Conference on Artificial Intelligence. Austin, TX. 322-325.

Winston, M.E., Chaffin, R., and Herrmann, D. (1987). "A Taxonomy of Part-whole Relations." *Cognitive Science* 11: 417-444.

Wittgenstein, L. (1953). Philosophical Investigations. Oxford: Basil Blackwell.

Zachman, J. (1987). "A Framework for Information Systems Architecture." *IBM Systems Journal* 26(3): 276-292.

Glossary

Analyst
: A primary developer of an ontology, in the context of an IDEF5 project. The central method for acquiring such information consists of interviewing domain experts and analyzing relevant domain documents.

Antisymmetry
: A property that holds of a given relation R when, for any objects x and y (of the appropriate kinds), if x bears R to y and y bears R to x, it follows that x is identical to y.

Arity
: The number of arguments of a relation, that is, the number of things that are involved in the relation in a given instance. For example, the relation *larger-than* has arity two, the relation *between* has arity three, and so forth.

Asymmetry
: A property that holds of a given relation R when, for any objects x and y (of the appropriate kinds), if x bears R to y, then y cannot bear R to x.

Attribute
: A *function*, or mapping, that takes each member of a given set of individuals to a single specific value. So, for example, the attribute *color-of* maps each object in a given set to its color; the attribute *age-of* maps each employee to his or her age.

Axiom
: A precise characterization of the logic of a term or set of related terms stated in the IDEF5 elaboration language. An axiom typically expresses a constraint on the objects denoted by the terms in axiom.

Characteristic
: A distinguishable feature. This term encompasses both attributes and properties. See **attributes** and **properties**.

Classification
: When applied to relations, categorizing objects in the domain of discourse according to how individuals relate to kinds and how kinds relate to one another in terms of subsumption and inclusion.

Client
: An individual or organization acting as a contracting agent for IDEF5 project services with an external source of IDEF5 expertise.

Commentor
: Member of an IDEF5 project team responsible for reviewing draft IDEF5 descriptions and making written critiques.

Constant
: A word in the IDEF5 elaboration language that is neither an operator nor a variable. Semantically, constants are words that denote objects in the domain.

Constraint	Most generally, a statement which must (or equivalently, must not) hold in a system. Most often, constraints express logical properties of, or connections between, domain objects that must be maintained if the system is to function as intended.
Context	A statement that identifies the boundaries of the IDEF5 project and the level of detail. The statement of context is important because it indicates the scope and level of granularity of the study.
Declarative Knowledge	The type of knowledge that human beings (domain experts) are aware of.
Definition	An expression that formally identifies an individual, relation, or function.
Domain	A sphere of interest, such as the semiconductor domain or the domain of abstract algebra. A domain has its own distinctive vocabulary for talking about the characteristic kinds of objects and processes typically found in the domain.
Domain Expert	An individual considered knowledgeable of, and conversant in, most of the distinguishing characteristics of a certain aspect of a domain. A role played by the primary sources of knowledge from the application domain of interest. Persons filling this role provide insights about the characteristics of the application domain that are needed for extracting the underlying ontological knowledge.
Elaboration Language	A structured textual language designed specifically to express ontology information. The IDEF5 elaboration language has the full power of first-order modal logic plus set theory.
Extensional	Criteria that specifies the identity of an abstract object be determined by its members or instances. Sets and, in some theories, classes, are the paradigmatic extensional entities: the principle of extensionality in set theory is that two purported sets are identical if and only if they have exactly the same members.
Form, Description Summary	A form that summarizes the evolving/completed ontology description. It records the purpose, viewpoint, and context and also provides a summary of all the schematics and documents used to record the domain ontology.
Form, Kind Specification	A form that records the information associated with a kind in a domain, in particular, the defining properties of the kind, relevant non-defining properties, relations the kind participates in, and other information.
Form, Proto-kind Specification	A form that records information associated with a proto-kind in a domain.

Form, Proto-relation Specification	A form that records information associated with a proto-relation in a domain.
Form, Relation Specification	A form that records information associated with a relation in a domain.
Form, Source Material Description	A form that records information associated with a source material.
Form, Source Statement Description	A form that records information associated with a source description.
Form, Term Description	A form that records list of the terms used to derive the ontology and a brief description of each term.
IDEF	Acronym for Integration Definition. Also used to refer to a family of mutually-supportive methods for enterprise integration including in particular IDEFØ, IDEF1, IDEF1X, IDEF3, IDEF4, and IDEF5.
IDEFØ	Integration Definition (IDEF) method for Function Modeling
IDEF1	Integration Definition (IDEF) method for Information Modeling
IDEF1X	Integration Definition (IDEF) method for Semantic Data Modeling
IDEF2	Integration Definition (IDEF) method for Simulation Modeling
IDEF3	Integration Definition (IDEF) method for Process Description Capture
IDEF4	Integration Definition (IDEF) method for Object-Oriented Design
Individual	The most logically basic kind of real world object. Prominent examples include human persons, concrete physical objects, and certain abstract objects such as programs. Unlike objects of higher logical orders such as properties and relations, individuals essentially are not multiply instantiable. Individuals are also known as *first-order* objects.
Initial Scope	A specification of the boundaries of the ontology development effort, in particularly, the parts of the systems that need to be included and those which are to be excluded from the ontological development effort.

Intensionality

In the context of IDEF5, the characteristic possessed by an abstract object when its identity is not determined by its members or instances. Properties are the paradigmatic intensional entities. Two distinct properties can have precisely the same instances without, intuitively, being identical. The property *being a living former U.S. president from California* and the property of *being the most famous actor-turned-politician* are intuitively distinct properties, yet have precisely the same instances, viz., Ronald Reagan.

Interview

A face-to-face meeting with domain experts for the purpose of pursuing some line of investigation.

Irreflexivity

The property that holds for a relation R if and only if no object stands in the relation R with itself.

Keyword

A word that has special meaning and usage in the IDEF5 elaboration language and cannot be used to denote objects in the domain of discourse.

Kind

Informally, a group of individuals that share some set of distinguished characteristics. More formally, kinds are properties typically expressed by common nouns such as 'employee', 'machine', and 'lathe'.

Kind Refinement Procedure

The activity of eliminating unvalidated proto-kinds from an ontology and promoting validated proto-kinds to kinds.

Knowledge Engineer

A technical role filled by personnel with IDEF5 expertise who are the primary developers of an IDEF5 ontology. Also known as an *analyst*.

Lexicon

The set of basic symbols of a language.

Meronymic Relations

Part-whole relations of various sorts, as, for instance, between an engine and the automobile that contains it, an acre of real estate and a larger piece of land containing it, the hydrogen in a cup of water and the water itself, and so forth.

Metaphysics

The branch of philosophy that systematically investigates first principles and, in particular, what ultimately exists.

Method

An organized, single-purpose discipline or practice for accomplishing some set of tasks. The IDEF methods are specifically designed to accelerate the learning process and help novice practitioners emulate the performance of highly experienced individuals engaged in a particular analysis or design activity. IDEF methods guide users through a disciplined approach, consistent with good-practice experience, to achieve consistently high levels of performance (quality and productivity).

Movement Protocol Analysis

A type of analysis in which idle movements are identified by studying motion efficiency.

Multiply Instantiable — Ability to have more than one instance or member. Sets, properties, relations, classes, types, and kinds are all examples of multiply instantiable objects.

Mutually Exclusive — A condition that exists, given M objects and N relations, when either the objects stand in none of the N relations or they stand in exactly one relation.

Object — In general, anything that can be referred to within a domain, including concrete and abstract things, as well as kinds, properties, and relations.

Object, First-Order — See **individual.**

Ontology — A domain vocabulary together with a set of precise definitions, or *axioms*, that constrain the meanings of the terms in that vocabulary sufficiently to enable consistent interpretation of data that use the vocabulary.

Ontology, Domain — The highest level of three distinguished levels of ontologies, when categorized in terms of generality. A domain ontology classifies the most general information that characterizes an entire domain. See **practice ontology** and **site-specific ontology**.

Ontology, Practice — An extension of a domain ontology that includes the common features of similar sites in that domain.

Ontology, Site-specific — An extension of a practice ontology (hence also a *domain ontology*) to include information about all of the relevant kinds of objects, properties, and relationships found within a specific site.

Operator — A lexical item in the IDEF5 elaboration language that attaches to terms to form other terms, or to statements to form other statements. The boolean operator "`not`" is a one-place sentence operator.

Pool, Kind — The collection of kinds that have been identified in an ontology.

Pool, Property — The collection of properties that have been identified in an ontology.

Pool, Proto-kind — The collection of proto-kinds that have been identified in an ontology.

Pool, Proto-characteristic — The collection of proto-characteristics that have been identified in an ontology.

Pool, Proto-relation — The collection of proto-relations that have been identified in an ontology.

Pool, Relation — The collection of relations that have been identified in an ontology.

Pool, Source Statement — The collection of meaningful statements made by different individuals, as well as statements extracted from source documents during the ontology development effort. Each source statement is given a unique identification number to improve traceability.

Pool, Term — The collection of meaningful terms used in an ontology. Typically these are the terms referring to proto-kinds, proto-properties, proto-relations, kinds, properties, and relations in the ontology.

Procedural Knowledge — Knowledge concerning the manner in which a certain task is carried out.

Process — A real world event or state of affairs involving one or more individuals over some (possibly instantaneous) interval of time. Typically, a process involves some sort of change in the properties of one or more of the individuals within the process. Because of the ambiguity in the term "process", sometimes referred to as *process instance*.

Process Kind — The abstract general character that is shared by similar processes. Such processes are said to be *instances* of the process kind. For example, the process kind *manufacture part* found in a certain enterprise is the general behavior exhibited by each particular instance in which a part is manufactured.

Project — A plan for conducting an IDEF5 ontology description capture effort with a clearly defined statement of purpose, context (scope and level of detail), and viewpoint.

Project Leader — An administrative role that carries the responsibilities for overseeing and guiding an ontology development effort. In particular, the project leader is ultimately responsible for the outcome of the ontology development effort, team organization and leadership, and schedule and budget management.

Prompting Questions — A question intended to prompt a domain expert to verbalize thoughts and/or to help guide a discussion.

Property — An abstract, general feature or characteristic that is multiply instantiable; that is, it can be shared by distinct objects.

Property, Accidental — A property that an individual has, but could have lacked.

Property, Defining — An element of the set of properties associated with membership in a given kind K.

Property, Essential — A property that an individual could not have failed to exemplify.

Property, First-Order — A property that holds only of individuals.

Property, Non-defining — A property which is important for characterizing a given kind, but which is not used in defining the kind, and is therefore not counted among the defining properties of the kind.

Property, Second-Order — A property that holds only of kinds and other first-order properties and first-order relations.

Protocol — An underlying pattern or structure of a discourse or behavioral process. The term protocol implies that an expert is solving a problem using commonly used approaches and tools.

Protocol Analysis — The process of analyzing a record of discourse or behavioral process. There are two types of protocol analysis: verbal protocol analysis and movement protocol analysis [Jackson, 90].

Proto-association Chart — A two dimensional matrix with relevant proto-kinds listed on both the axes. An X is marked in cells to indicate the possible existence of a proto-relation.

Proto-characteristic — A characteristic tentatively identified for inclusion in an ontology. A proto-characteristic at a later point is either eliminated from the developing ontology or elevated to the status of a full fledged property or attribute. Tentatively identified relations are known as proto-relations, and have the same status as proto-characteristics.

Proto-kind — A group in a domain tentatively identified as a kind. A proto-kind at a later point is either eliminated from the developing ontology or elevated to the status of a full fledged kind.

Proto-relation — See **proto-characteristic**.

Reader — A member of an IDEF5 project team responsible for reviewing draft IDEF5 descriptions but who is not responsible for providing written comments.

Referent — A construct in the IDEF5 elaboration language used to refer to a kind, object, property, relation, or process kind in another ontology or an IDEF model.

Reflexivity — A property that holds of a given relation R if, for any object x (of the appropriate kind), x bears the relation R to itself. An example of a reflexive relation is *weighs-as-much-as*.

Relation — An abstract, general association or connection that holds between two or more objects. Like properties, relations are multiply instantiable. The objects among which a relation holds in a particular instance are known as its *arguments*.

Relation, First-Order — A relation that can hold only between individuals.

Relation, Second-Order — A relation, one of whose arguments is a kind, property, or first-order relation.

Relation, Spatial — A relation between spatial locations such as *to the left of*, *above*, *contiguous with* and so forth.

Relation, Temporal — A relation between temporal points or intervals such as *before*, *during*, *overlaps* and so forth.

Relation Library — A collection of common, predefined, axiomatized relations that are available to IDEF5 users. The library is extensible.

Relation Refinement Procedure — The activity of eliminating unvalidated proto-relations from an ontology and promoting validated proto-relations to relations.

Reviewer — A member of an IDEF5 project team who is knowledgeable about the application domain and/or the IDEF5 method and is responsible for reviewing and commenting on draft descriptions and documents. Team members and domain experts can be reviewers. See also *reader* and *commentor*.

Sanctioned Inference — See **constraint**.

Schematic — A connected diagram constructed from the lexicon of the IDEF5 schematic language, in accordance with the syntactic guidelines of the language.

Schematic, Basic First-Order — Either an existential schematic, or an *n*-place first-order schematic.

Schematic, Basic Second-Order — A schematic consisting of two kind symbols, or two relation symbols, or a kind symbol and a relation symbol connected by a single second-order relation symbol.

Schematic, Classification — A schematic representing how the *subkind-of* relation holds between different kinds in a domain.

Schematic, Complex First-Order — Any first-order schematic other than a basic first-order schematic.

Schematic, Composition — A schematic representing how the *part-of* relation holds between the instances of different kinds in a domain.

Schematics, Existential — A schematic consisting of a single individual, kind, or relation symbol. Such schematics enable a domain expert to record the mere fact that certain indiviuduals, kinds, or relations have been observed in a given domain without requiring any further information about the relations such objects stand in with other objects in the domain.

Schematic, First-Order — Either a basic first-order schematic, or the result of connecting a kind symbol in a given first-order schematic to another kind symbol by a first-order relation symbol.

Schematic, Object-State — The basic construct for describing process kinds in the IDEF5 schematic language.

Schematic, Relation — A schematic consisting only of first-order relation symbols connected by second-order relation symbols.

Schematic, Second-Order — A schematic involving at least one second-order relation symbol.

Schematic, State Composition — A complex first-order schematic used to represent how objects of certain kinds are transformed to yield an object of some kind.

Schematic, *n*-place — For $n \geq 1$, a schematic consisting of n kind and individual symbols connected by a single n-place first-order relation symbol.

Schematic Language, IDEF5 — The graphical component of the IDEF5 languages.

Sentence — A sentence in the IDEF5 elaboration language is an expression of some fact that is observed or believed to be true in a domain.

Source Material — A textbook, a research article, an enterprise-specific document such as a policy manual or a procedure manual, a set of an interview notes, or direct observation notes that has relevant information to the ontology development project.

Source Material Log — A document which serves as the primary index to all source material collected and used in an IDEF5 project.

State — A property, generally indicated by an adjective rather than a common noun, that is characteristic of objects of a certain kind at a certain point within a process. For example, water can be in frozen, liquid, or gaseous states.

Statement of Need (SON) — A statement that records the source of the request (person or project) and paraphrases the stated objectives of the project.

Statement of Purpose — A statement that clearly specifies the main objective(s) that the ontology development team intends to achieve. Defining the purpose can be separated into two parts, 1) defining a statement of need (SON) and 2) defining the information goals in terms of how the ontology will be used.

Subkind-of — The characteristic relation that holds between a given kind and its subkinds. For example, the kinds **capstan lathe** and **turret lathe** bear the *subkind-of* relation to **lathe**.

Symmetry — A property that holds of a given relation R if, for any objects x and y (of the appropriate kinds), if x bears the relation R to y, then y bears R to x. An example of a symmetric relation is *contiguous-with.*

System	A collection of physical and/or conceptual objects that work together to achieve common objective.
Team Member	A person involved with the IDEF5 ontology description project.
Term Coining	The strategy of coining a term for new a new domain object.
Transitivity	A property that holds of a given relation R if, for any objects x, y, and z (of the appropriate kinds), if x bears the relation R to y, and y bears R to z, then x bears R to z. An example of a transitive relation is *larger-than*.
Verbal Protocol Analysis	A method of acquiring domain knowledge in which experts are asked to think aloud during a problem solving activity. The data derived from this exercise are then analyzed, and domain knowledge extracted.
Variable	A term in the IDEF5 elaboration language that *ranges over*, i.e., can take arbitrary semantic values in, some given domain of objects.
Variable, Individual	A variable that ranges over the individuals in a domain.
Variable, Second-Order	A variable that ranges over kinds and first-order relations in a domain.
Variable, Sequence	A variable that ranges over finite sequences of individuals in a domain.